1992-12

Microcomputer simulation of a Fourier approach to ultrasonic wave propagation.

Reid, William H.

Monterey, California. Naval Postgraduate School

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Microcomputer Simulation of a Fourier Approach to Optical Wave Propagation

12. PERSONAL AUTHOR(S)
REID, William H.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year,Month,Day)	15. PAGE COUNT
Master's Thesis	FROM _____ TO _____	December 1992	105

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
			2-D spatial Fourier transform, Green's function, spatial impulse response,diffraction, pulsed waves, ultrasonic propagation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis uses linear systems approach and the Fourier transform as the basis for a microcomputer program to model pulsed ultrasonic wave propagation. The program computes the acoustic potential in a plane at a given distance from the source. The mathematical development establishes the importance of the total impulse response as the Green's function, meeting the boundary conditions and solving the wave equation. Four excitation functions are presented. The square and circular piston excitations are used to verify the program. Excitation functions also modeled are the circularly truncated Gaussian distribution and the circularly truncated Bessel profile. All programs were written using the MATLAB software package. This work provides a computationally efficient means to analyze pulsed ultrasonic wave propagation as a spatially filtered source.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL POWERS, John P.	22b. TELEPHONE (Include Area Code) (408) 646 - 2679	22c. OFFICE SYMBOL EC/Po

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE
S/N 0102-LF-014-6603 Unclassified

Approved for public release; distribution is unlimited.

MICROCOMPUTER SIMULATION OF A FOURIER APPROACH TO ULTRASONIC WAVE PROPAGATION

by

William H. Reid
Lieutenant, United States Navy
B.S., Virginia Military Institute, 1985

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

ABSTRACT

This thesis uses a linear systems approach and the Fourier transform as the basis for a microcomputer program to model pulsed ultrasonic wave propagation. The program computes the acoustic potential in a plane at a given distance from the source. The mathematical development establishes the importance of the total impulse response as the Green's function, meeting the boundary conditions and solving the wave equation. Four excitation functions are presented. The square and circular piston excitations are used to verify the program. Excitation functions also modeled are the circularly truncated Gaussian distribution and the circularly truncated Bessel profile. All programs were written using the MATLAB software package. This work provides a computationally efficient means to analyze pulsed ultrasonic wave propagation as a spatially filtered source.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

The propagation characteristics of continuously radiated monochromatic ultrasonic sources are well solved through application of the angular spectrum technique [Ref. 1] or Fresnel integrals. More frequently, however, acoustic imaging, tissue characterization, and physical acoustics applications tend to use pulsed sound. The propagation of pulsed ultrasound with arbitrary temporal and spatial components is not understood to the same degree. A path to greater understanding is the development of a reliable, easy method of diffraction prediction. This thesis examines an approach based on linear systems theory and the Fourier transform. The thesis goal was to achieve a readily usable method of predicting pulsed wave diffraction in a time-efficient and accurate manner in order to examine the wave diffraction.

The basic method of the spatial impulse response was introduced by Stepanishen [Refs. 2–5], reviewed by Harris [Ref. 6], and modified by Guyomar and Powers [Refs. 7–10]. The Guyomar/Powers approach differed by the use of linear systems theory. Linear systems theory revealed the importance of the total impulse response and its equivalence to the Green's function. Furthermore, the spatial impulse response functions are found in the spatial transform (Fourier) domain. Working in the transform domain allowed propagation of the wave to be viewed as a time-varying spatial filter applied to the spatial spectrum of the input excitation.

The advantage of this method is that it provides the diffracted field from an insonifying wave with arbitrary temporal and spatial dependence in a computationally efficient form. By use of the Fourier transform, an amenable computer implementation of this technique using FFT routines is possible.

The desired benefit of a fast, time-efficient computer implementation to calculate the acoustic potential or pressure is to aid in ultrasonic transducer design for medical, acoustic imaging, and mine warfare applications. With a knowledge of wave diffraction phenomenon a diffracted wave reflected from an unknown object can be used to provide information about the object. This type of system must be portable as well as time-efficient, which is very achievable given the trends in computer technology. Computers have become faster and have increased memory capacity while their size has decreased. Other benefits are derived from the use of the matrix manipulation program, MATLAB, and the ability to expand this implementation to cases involving lossy media. Because MATLAB is readily available on the commercial market, it requires no special equipment for computer implementation.

Previous computer implementations have been Fortran coded [Ref. 11] to be run on mainframes and microcomputers. The specific goal of the microcomputer implementation by Merrill [Ref. 11] was a run time under 30 minutes; this goal was achieved. Merrill first attempted to use MATLAB, but failed due to limitations on computer memory and MATLAB array size. The present thesis work focused on implementation of the method via MATLAB code, the addition of a Bessel profile

excitation, and a comparison between the Gaussian wave and the Bessel wave. Further study of these excitations appears in Chapter IV.

Chapter II consists of the problem description, including the source-to-observation plane geometry, a discussion on the linear systems approach, the mathematical development, and an overview of MATLAB and AXUM. Chapter III consists of a discussion of the two program modules, the acoustic filter module, AC_FIL.M, and the acoustic propagation module, AC_PROP.M. Chapter IV starts with the set of defining parameters. These parameters are then used for an explanation of the program's verification and an investigation of other input excitation functions. Following a summary in Chapter V, Appendices A and D give detailed explanations of AC_FIL.M and AC_PROP.M, respectively. The source code follows each respective explanation in Appendices B and E. Appendix C gives examples of the filters generated by the code in Appendix B. The source code of the excitation functions is given in Appendix F. Examples of the outputs for **Gaussian** and **Bessel** inputs are given in Appendices G and H, respectively.

II. PROBLEM DESCRIPTION

Before assembling a computer implementation, we must first understand the problem. What follows here is an explanation of the problem beginning with the description of the geometry in the first section. Section two continues the explanation into the linear systems approach. The third section proceeds through the mathematical development of the problem and ties in the Fourier transform. The theory presented in the first three sections was derived from the works of Guyomar and Powers [Refs. 7, 8, 9, and 10]. The final section gives an overview of the tools used for generating the inputs, outputs, and output graphics.

A. GEOMETRY

The problem geometry is shown in Fig. 1. The acoustic velocity potential $\phi(x,y,z)$ is to be calculated at an arbitrary point in the positive-z half-space given a z-directed velocity in a portion of the source plane. The source's z-directed velocity is spatially and temporally arbitrary and is rigidly baffled (i.e., equal to zero) in the region outside the source. Furthermore, it is assumed that the spatial and temporal components of the z velocity are separable, having the form at the input plane

$$v_z(x,y,0,t) = T(t)s(x,y). \tag{1}$$

A linear, homogeneous, and lossless (in this case) medium is assumed to be present between the source and observation point. [Refs. 7, 8, 9, and 10]

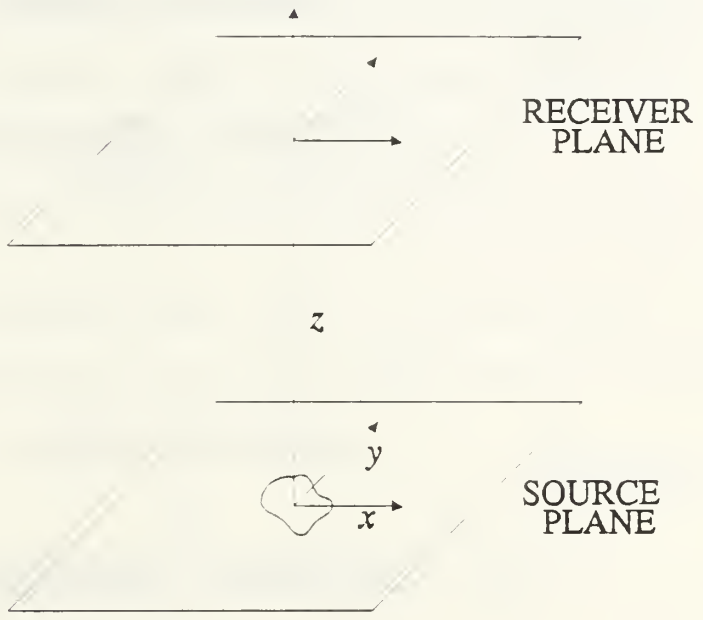

Figure 1. Source-to-receiver geometry.

B. LINEAR SYSTEMS APPROACH

Linear systems solutions are applied to systems that are linear and time-invariant. A linear systems solution approach to this problem is possible because propagation in a linear homogeneous media is a linear, space-invariant process [Ref. 7]. In linear systems the impulse response is the response of the system to an impulsive input. The *total impulse response* $h(x,y,z,t)$ of a system is produced by an input of the form $\delta(x,y)\delta(t)$; this is shown in Fig. 2(a). Figure 2(b) shows the *spatial impulse response* $p(x,y,z,t)$, defined as the response to an excitation of the form

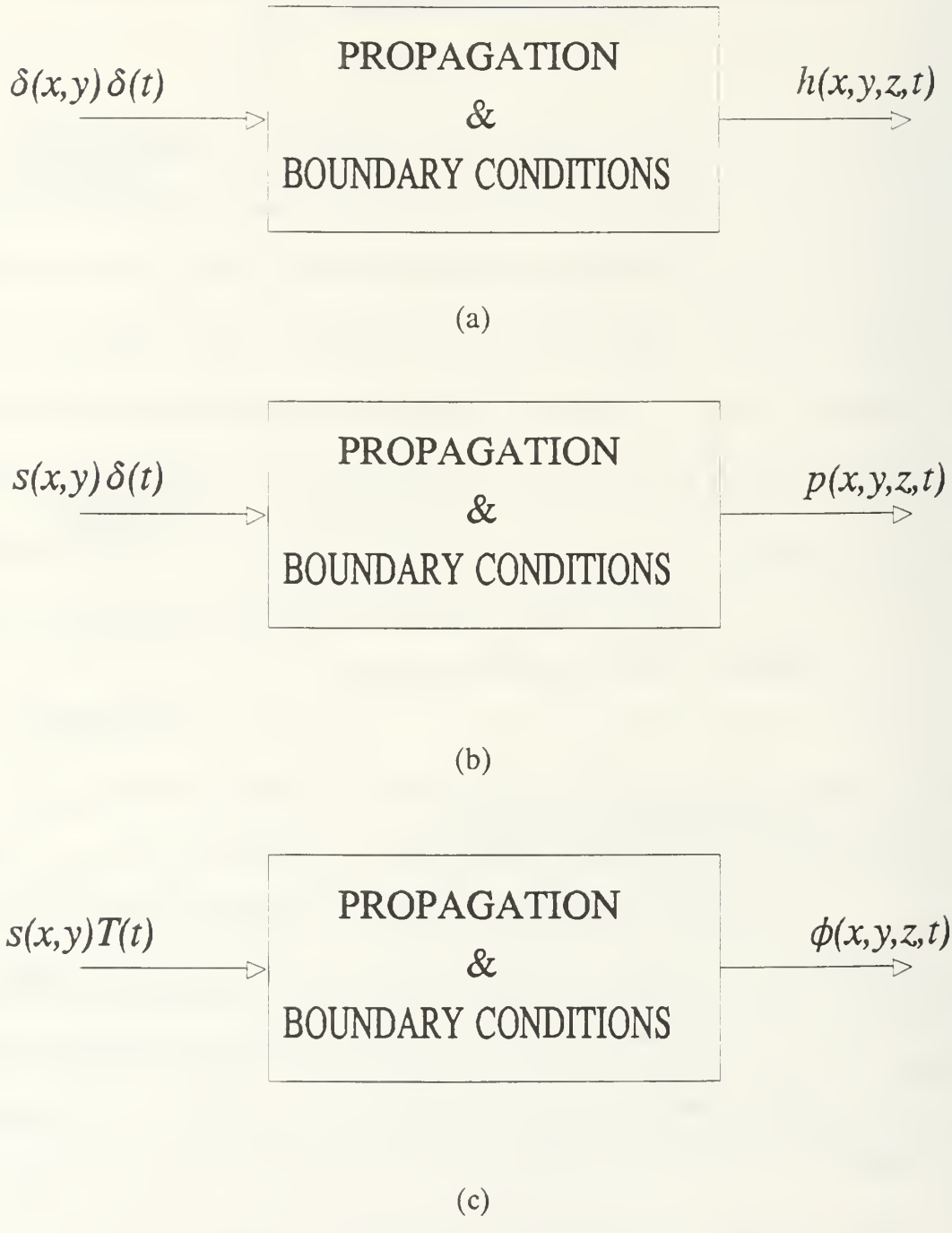

Figure 2. Block diagram of the (a) total impulse response, (b) spatial impulse response, and (c) general solution.

$s(x,y)\,\delta(t)$. Note that an arbitrary spatial input has been substituted for the impulsive spatial input. Recall from linear systems theory that the solution for an arbitrary input (spatial of temporal) is the convolution of the input with the system's total impulse response; therefore, the spatial impulse response in this case is given by

$$p(x,y,z,t) = s(x,y) \overset{*}{x} \overset{*}{y} h(x,y,z,t) \tag{2}$$

where * indicates convolution with respect to the variable shown.

The general solution has a linear systems representation, as shown in Fig. 2(c), where

$$\phi(x,y,z,t) = s(x,y)T(t) \overset{*}{x} \overset{*}{y} \overset{*}{t} h(x,y,z,t). \tag{3}$$

Substituting Eq. (2) into Eq. (3) gives

$$\phi(x,y,z,t) = T(t) \overset{*}{t} p(x,y,z,t). \tag{4}$$

It follows from Fig. 2(c) and Eqs. (2) and (4) that the key to the solution is finding the spatial impulse response $p(x,y,z,t)$ which is, itself, dependent on the total impulse response of the system $h(x,y,z,t)$. The total impulse response of the system is the propagation field that results from an impulsive source, as in Fig. 2(a), that solves the wave equation and satisfies the boundary conditions. The solution to the wave equation satisfying the boundary conditions is commonly known as the Green's function; hence, the total impulse response is simply the Green's function. Therefore, once the Green's function is known, the total impulse response is also known, and

the problem becomes a triple convolution between an excitation source which is spatially and temporally arbitrary (and assumed to be known), and the systems' total impulse response or Green's function. [Refs. 7, 9, and 10]

C. MATHEMATICAL DEVELOPMENT

The wave equation for lossless, rigidly baffled media solved by Green's function is

$$\nabla^2\phi - \frac{1}{c^2}\frac{\partial^2\phi}{\partial t^2} = 0. \tag{5}$$

The general solution of Fig. 2(c) gives the result in terms of the acoustic potential ϕ which must be found from a z velocity input. To relate acoustic velocity to acoustic potential the following relationship is used

$$v(x,y,z,t) = -\nabla\phi(x,y,z,t) \tag{6}$$

resulting in the z velocity component given by

$$v_z(x,y,z,t) = -\frac{\partial\phi(x,y,z,t)}{\partial z}. \tag{7}$$

Since the wave equation is in terms of c (the acoustic velocity in the media) and time t, the partial derivative with respect to z must be related to these two parameters. This is done by using the fact that a wave traveling in the positive-z direction has an argument of the form $ct-z$, resulting in the relationship

$$\frac{\partial}{\partial z} = -c\frac{\partial}{\partial t}. \tag{8}$$

Applying Eq. (8) to Eq. (7) gives the z velocity at the input plane z=0 as

$$v_z(x,y,z,t) = c\frac{\partial\phi(x,y,0,t)}{\partial t}. \tag{9}$$

Equation (9) requires the acoustic potential at the input plane. It was assumed in Eq. (1) that the z velocity is separable which implies a separable acoustic potential. Such an acoustic potential has the form

$$\phi(x,y,0,t) = s(x,y)\tau(t) \tag{10}$$

at the z=0 plane. If Eq. (10) is substituted into Eq. (9) and the partial derivative carried out, then Eq. (9) becomes

$$v_z(x,y,z,t) = cs(x,y)\tau'(t). \tag{11}$$

A comparison of the z velocities given in Eqs. (1) and (11) indicates that $T(t)$ is equivalent to $c\tau'(t)$ where $\tau'(t)$ is the time derivative of the time component of the acoustic potential given in Eq. (10). Equation (11) is now the input in Fig. 2(c).

As stated earlier, the general solution of Fig. 2(c) is solved by the Green's function. For the standard wave equation (for lossless media), Eq. (5), and rigidly baffled boundary conditions the applicable Green's function is

$$h(x,y,z,t) = \frac{\delta(ct-z)}{2\pi R} \tag{12}$$

where $R = \sqrt{x^2+y^2+z^2}$ [Ref. 9]. Substituting this in Eq. (2) provides the spatial impulse response

$$\begin{aligned} p(x,y,z,t) &= s(x,y) \overset{*}{x} \overset{*}{y} h(x,y,z,t) \\ &= s(x,y) \overset{*}{x} \overset{*}{y} \frac{\delta(ct-t)}{2\pi R}. \end{aligned} \tag{13}$$

Of course, the substitution will trickle through Eqs. (3) and (4).

The double convolution in Eq. (13) is arduous to compute and not easily computer implemented. The Fourier transform furnishes a convenient method to resolve this dilemma by using the Fourier property that convolution in the spatial domain becomes multiplication in the transform domain. Applying this property to the relationship depicted in Fig. 2(b) and taking the two-dimensional spatial Fourier transform of Eq. (13) results in the relationship shown in Fig. 3.

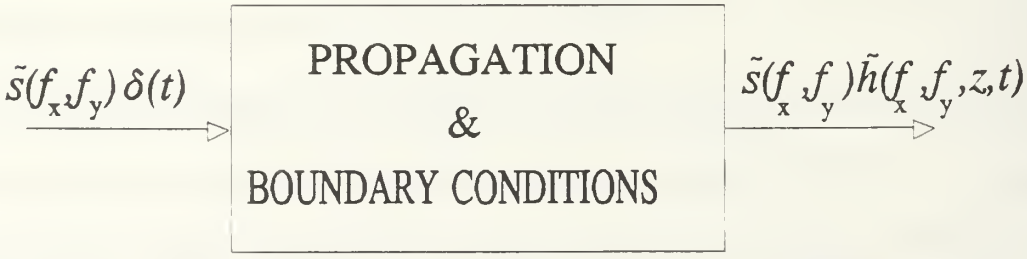

Figure 3. The spatial impulse response in the Fourier transform domain.

Thus the spatial impulse response in the spatial transfer domain is the product of the angular spectrum of the source $\tilde{s}$ and the propagation transfer function $\tilde{h}$, written symbolically as

$$\tilde{p}(f_x f_y z,t) = \tilde{s}(f_x f_y)\tilde{h}(f_x f_y z,t). \tag{14}$$

Here the Green's function has been substituted in as the total impulse response. Taking the two-dimensional spatial Fourier transform of the Green's function in Eq. (12) gives the propagation transfer function

$$\begin{aligned} \tilde{h}(f_x f_y z,t) &= \mathcal{F}\left\{\frac{\delta(ct-t)}{2\pi R}\right\} \\ &= 2J_0\left(\rho\sqrt{c^2t^2-z^2}\right)H(ct-z) \end{aligned} \tag{15}$$

where the relationship

$$\mathcal{F}\left\{\frac{\delta\left(t-\frac{R}{c}\right)}{R^n}\right\} = \frac{J_o\left(\rho\sqrt{c^2t^2-z^2}\right)}{(ct)^{n-1}} \tag{16}$$

was applied where $\rho = \sqrt{f_x^2+f_y^2}$. The term $H(ct-z)$ is the step function.

Equation (15) exhibits two important points. The first point is that the propagation transfer function is a Bessel of the first kind of zero order. Secondly, the propagation transfer function can be identified as a time-varying spatial filter having a Bessel shape. These filters are produced by AC_FIL.M.

Equation (13) is for an input that is temporally impulsive and spatially arbitrary. Taking the inverse two-dimensional spatial transform of Eq. (13) would, in

this case, give a final result since convolution with an impulse is the same function at the time that the impulse occurred. To account for a non-impulsive time input component, the convolution of Eq. (4) must be carried out; Eq. (17) gives the solution,

$$\phi(x,y,z,t) = T(t) \ddagger \mathcal{F}^{-1}\{\tilde{p}(f_x,f_y,z,t)\} \tag{17}$$

where $\tilde{p}$ is the product of the angular spectrum of the source $\tilde{s}$ and the propagation transfer function $\tilde{h}$. Since only temporally impulsive inputs are simulated in the examples that follow, the convolution in Eq. (17) was not computed for our cases. Our final solution, therefore, reduces to taking the inverse 2-D spatial Fourier transform of the spatial impulse response $\tilde{p}$. The thesis simulates the spatial impulse response solution for the input excitation function distribution chosen by the user in the program module AC_PROP.M.

The program that implements these equations is discussed in Chapter III and detailed in Appendices B and D. Illustrative examples of the time-varying filters and outputs follow in the Chapter IV discussion with more examples supplied in Appendices C, G, and H. The tool of implementation for the program was MATLAB with graphical assistance provided by AXUM. An overview of both follows in the next section.

D. TOOLS EMPLOYED

1. MATLAB OVERVIEW

MATLAB is a high-performance, interactive, scientific and engineering numeric computation software package. The name comes from MATrix LABoratory; hence, the basic data element is a matrix which does not require dimensioning. MATLAB, however, has evolved into a versatile scientific "spreadsheet" for numeric calculations. A major advantage of MATLAB is that traditional programming is not needed since problems and solutions are expressed just as they would be written mathematically. Another distinct advantage is MATLAB's expansion capability through the use of preprogramed functions, such as calculation of the two-dimensional FFTs and Bessel function. [Ref. 12]

A function is one of two types of m-files (called m-files for the ".m" suffix); the other is a script file. Script files are used to automate long sequences of commands including functions. Arguments are not passed into script files. Functions, however, may have arguments passed into them. Another difference between the two file types is that the first line of a function file begins with the word "function" and all variables used in the function are local. Examples of script files in this thesis include AC_FIL.M and AC_PROP.M. Examples of functions include the input functions (see Appendix F), the three dimensional graphing function ***mesh***, and the two "fft" functions that realize the Fourier transform. [Ref. 12]

The two "fft" functions employed for the Fourier transform are ***fft2*** and ***fftshift***. The ***fft2*** function is a two-dimensional fast Fourier transform and is

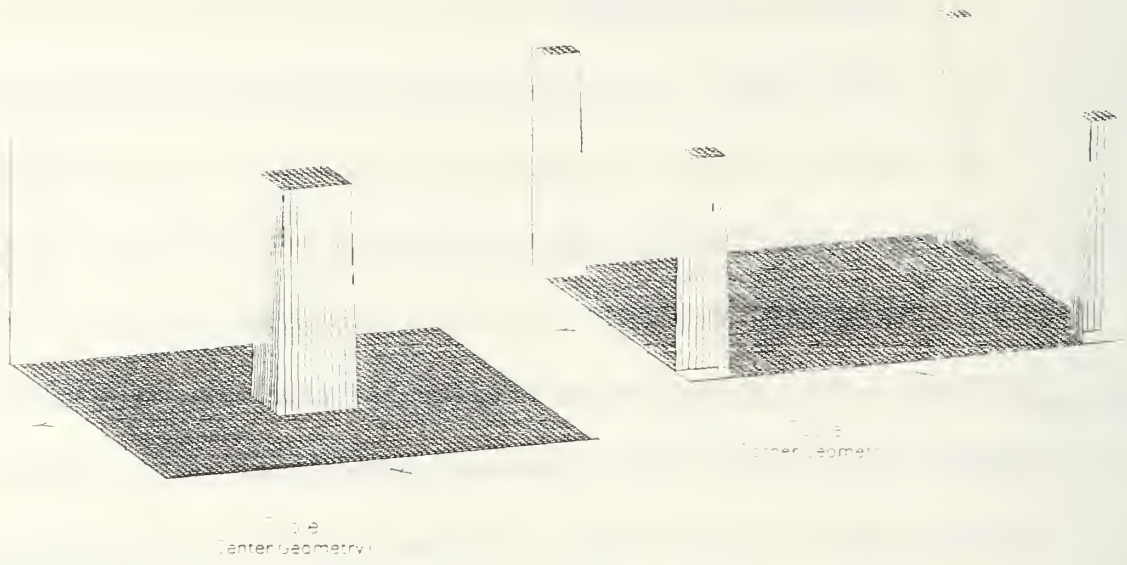

Figure 4. Illustration of the center versus corner geometry.

repetitively used throughout the program. Before the ***fft2*** operation is performed it is necessary to execute the ***fftshift*** function. Figure 4 shows a rectangular **Table** input before (left image) and after (right image) execution of ***fftshift***. The center geometry is convenient for viewing; however, a phase shift is introduced if the ***fft2*** operation is applied to this geometry because of the lateral translation of the object away from the origin. MATLAB, recognizing the introduced phase shift, solved the problem with the ***fftshift*** function. In the corner geometry (the right image of Fig. 4) MATLAB assumes replication on all sides, forming a function that is periodic in both the x and y directions. The assumed replication forms an identical function at the origin and hence, the correct phase is preserved through the ***fft2*** operation. In keeping with MATLAB terminology, the variables used to encode the preceeding equations that

are in the center geometry have the indicator "shft" appearing in their names; those in the corner geometry have no such indicator. The same logic applies to the inverse Fourier operation (*ifft2*). The interested reader is referred the MATLAB User's Guide [Ref. 12] for more details. Both geometries can be viewed using the *mesh* command.

Because MATLAB normalizes the plot display, its 3-D graphics using *mesh* are outstanding for comparing relative shapes. Unfortunately, MATLAB has no option to display 3-D graphics with axes. A forthcoming version of MATLAB, MATLAB4, addresses this weakness. MATLAB4, however, is not currently available commercially. Due to the three dimensional graphing weakness in the current version of MATLAB, another graphics package was used. The input and output data was formatted in ASCII by MATLAB and stored on disk. The data was then imported into the graphic package AXUM.

2. OVERVIEW OF AXUM

AXUM is an interactive software package for technical graphics and data analysis. Produced by TriMetrix Inc., AXUM allows easy manipulation of raw data imported from MATLAB in ASCII format (data may also be imported from several other formats). Once imported, data manipulation can be carried out by AXUM through use of the *Transform* and *Convert* menu options. Then the data is arranged under the *History* menu using the *MAT2GRID* routine, so that the desired surface plot can be generated. The *MAT2GRID* routine produces three columns of data from the original matrix by placing each element of acoustic potential data in the "z" data

column with the indices from the matrix in the "x" and "y" columns. Once processed, the data is ready for graphing. [Ref. 13]

The *Graph* menu gives various options for controlling the graph's attributes and general aesthetics. Once the desired axes intervals, labels, and titles have been set, the graph can be displayed in one of two windows, the *Edit Screen* or the *View* window. Whereas the *View* window is simply for viewing, the *Edit Screen* window allows changes to be made to the graph while it is displayed. (This is time consuming because the graph is redrawn after each change.) A useful feature of the *Edit Screen*, however, is the capability to rotate and tilt the graph interactively so that the preferred aspect is achieved. After establishing the desired attributes, the graph can be saved as a graph or as an image. [Ref. 13] The concept of saving the graphs as images is another very useful feature because it allows a single graph format to be assembled and saved as a graph template on which the images may be overlaid. The example figures of this thesis were created in this manner (see Appendices C, G, and H).

III. MATLAB MODELING OF EQUATIONS

This chapter discusses the two MATLAB script files, AC_FIL.M and AC_PROP.M. Together, the script files form the two modules of the program that simulates acoustic wave diffraction via implementation of the concepts and equations presented in the previous chapter. The program was written in two modules due to the time consuming computation of the Bessel functions for the filters. Modularity also allows the propagation module to be run for several different input functions without having to recalculate the filters. A working narrative of AC_FIL.M opens the discussion, followed by a working narrative of AC_PROP.M. The excitation functions, or input functions, will be included under the AC_PROP.M heading. A brief summary of the program steps follows in the final section.

A. ACOUSTIC FILTER MODULE

The script file AC_FIL.M for ACoustic FILter (referred to as AC_FIL hereafter) computes the time-varying Bessel filters of Eq. (15), repeated here for convenience as

$$\tilde{h}(f_x f_y z,t) = \mathcal{F}\left\{\frac{\delta(ct-t)}{2\pi R}\right\} = 2J_0\left(\rho\sqrt{c^2t^2-z^2}\right)H(ct-z)\ . \tag{18}$$

In the discussion that follows equation variables are in **bold** and the related MATLAB code variables are in *italics*. Before discussing the coding of Eq. (18), the basic setup must be explained.

The basic setup consists of setting the size of the array and the sampling frequency. The variable N denotes the number of points on a side in the base array creating an NxN matrix. Initially N was given a value of 64 as in previous work [Refs. 7-11]. Once the program result was verified, N was increased by a factor of two to 128. Note that both values of N are powers of two; this was done to make use MATLAB's faster radix-2 "fft" function. Making N an even number, however, caused the point of singular information, the center of symmetry, to fall between matrix elements. To force the center of symmetry *N0* to coincide with a matrix element, the center was located at

$$N0 = \left(\frac{N}{2}\right) + 1. \tag{19}$$

Equation (19) results in an offset geometry as shown in Fig. 5.

The offset center (*N0*,*N0*) divided the base array into four matrices, all with different sizes. The fact that the matrices have different sizes is important in the use of symmetry because only one quadrant of information is actually computed. We calculated the data for the $N0$x($N0-1$) points in Quadrant I. In addition data was calculated for an additional column ($N0$x1) located at the right side of Quadrant I. (The extra column was required to compute all of the values in the other three quadrants using symmetry.) Quadrants II, III, and IV were determined from

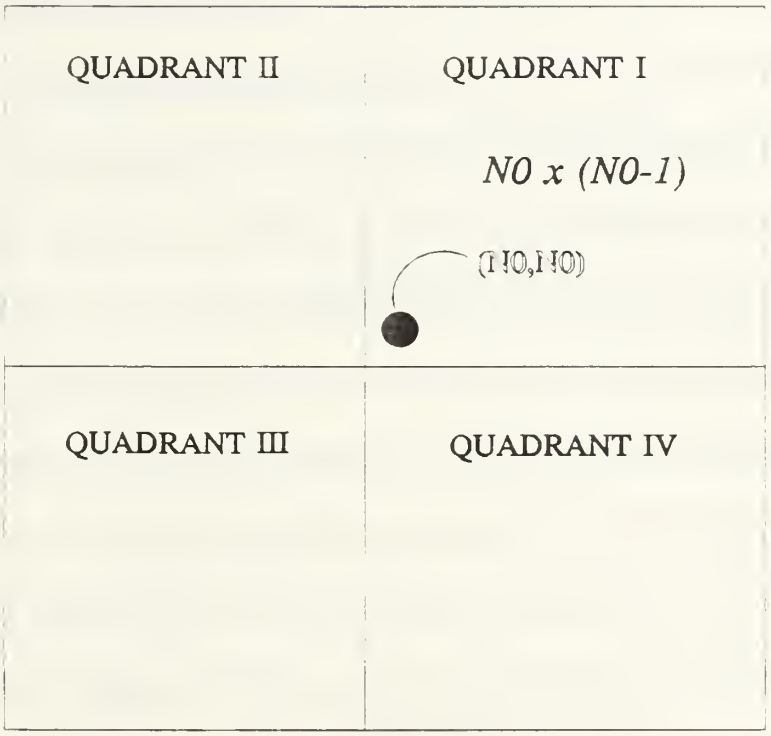

Figure 5. Offset geometry of base array matrix.

Quadrant I using symmetry with MATLAB's "flip" commands. The use of symmetry in this manner was employed in both program modules.

The number of time samples (or number of time slices), M, was set at 64. Initially, this was to emulate previous work [Refs. 7-11]; however, the resolution proved to be adequate for the follow-on simulations. Though there are 64 time slices, only 61 filters are generated by the MATLAB implementation of Eq. (18). The Heaviside step function in Eq. (18) was simulated by arbitrarily setting a variable *Step* to three which produces all-zero rows for the first three time slices. The result is that $M-Step$, or 61, time slices actually require computing. Since the first three

time slices are zero, the computations start with the fourth time slice at time z/c and proceed to the time value, *time_max*, provided by the user.

Referring back to Eq. (18), it is seen that the argument to the Bessel function that generates the time-varying spatial filter is composed of four variables. Of the four, only the time **t** varies within the program. In the MATLAB code time **t** is represented by the variable *time*. As previously stated, filter generation does not start until time z/c and is linearly incremented to the maximum time of propagation *time_max*. The source-to-receiver distance **z** was assigned the value $z = 10$ cm in the MATLAB code and **c**, the acoustic velocity in the medium, was assigned the value $c = 1500$ m/s. This value of c is the velocity of sound in water. The value of z was originally chosen to parallel previous work [Refs. 7-11] and was found to be convenient for subsequent simulations. The value of *time_max* was set to 150 μs for the verification phase and to 375 μs for the remainder of the simulations. Consequently, *time* ranged from 66.667 μs to 150 μs (or 375 μs) in 61 increments.

The final variable in the Bessel argument to be examined is **ρ**. In the MATLAB code **ρ** has the name *rho* and has a maximum value, *rho_max*, of 200. The value of *rho_max* = 200 was arbitrarily chosen; however, the 200 value chosen follows the work done by Merrill [Ref. 11]. Although *rho* is not time-varying, it does vary with spatial frequency,

$$\rho = \sqrt{f_x^2 + f_y^2}. \tag{20}$$

A vector having $N0 - 1$ points extending from 0 to *rho_max* was then formed. Then

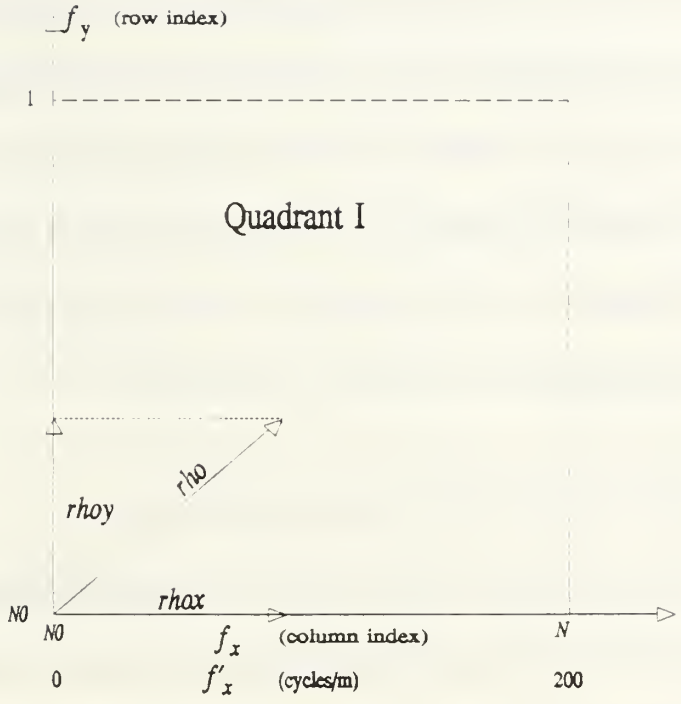

Figure 6. Construction of *rho* shown graphically.

the vector was used in the MATLAB routine ***meshdom*** [Ref. 12] to form two identical matrices *rhox* and *rhoy*. The ***meshdom*** routine takes the vector and forms a matrix with the same spacing in the x direction, *rhox*, and the same spacing in the y direction, *rhoy*. The two matrices *rhox* and *rhoy* represent f_x and f_y in Eq. (20) and Fig. 6 which shows graphically the construction of *rho*. In Fig. 6 f_x and f_y are in terms of the column and row numbers, respectively. The row index runs from 1 to *N0*; the column index runs from *N0* to *N*. The f_x axis can be rescaled to units of spatial frequency (f'_x) with units of cycles/m. This is represented in Fig. 6 by the f'_x axis labeling.

The combination of *rho* and the other variables forms the argument *arg* to the MATLAB Bessel function. Since *time* varies for each time slice, *arg* varies for each time slice generating a filter per time slice. After generation each filter is stored to disk for use by AC_PROP.M. The variables *N*, *N0*, *M*, and *Step* are also stored to disk for use by AC_PROP.M. The interested reader can find a detailed explanation and the source code in Appendices A and B, respectively. Graphical examples of the time-varying filters are include as Appendix C.

B. ACOUSTIC PROPAGATION MODULE

The script file AC_PROP.M for ACoustic PROPagation (referred to hereafter as AC_PROP) takes the user's chosen input function and simulates the propagation as a function of time. AC_PROP computes the spatial impulse response *p(x,y,z,t)* of Eq. (21)

$$p(x,y,z,t) = \mathcal{F}^{-1}\left\{\tilde{s}(f_x,f_y)\,\tilde{h}(f_x,f_y,z,t)\right\} \tag{21}$$

using the Green's function, or time-varying Bessel filters, produced by AC_FIL. In Eq. (21), $\mathcal{F}^{-1}$ is the inverse spatial Fourier transform operator and the ~ denotes a transfer domain quantity. To accomplish the computation of *p(x,y,z,t)*, AC_PROP first loads the variables passed from AC_FIL. Once complete, AC_PROP queries the user to make an input function choice.

The user is given four input functions from which to choose; these choices are **Circle**, **Table**, **Gaussian**, and **Bessel**. The **Table** and **Circle** are uniformly distributed

having the shape of a square and a circle, respectively (commonly known as the square and circular piston). The **Gaussian** and **Bessel** are truncated circular functions that have the indicated distribution within the circle. Pursuant to the input function choice, the user also is asked to input the circle diameter d (or square width w in the case of the **Table**); d (and w) were set to 11 for program verification. Once the program was verified, the diameter d was increased to 51, coinciding with the increase of N to 128. Examples for $N = 128$ and $d = 25$ are included in Apendices G and H. In the case of the **Gaussian** and **Bessel**, the user is further requested to input the standard deviation σ or a scaling factor a, respectively. These two function defining parameters were varied on a case by case basis. (Appendices G and H provide examples.) To review the input functions, see the source codes contained in Appendix F. The variable *shft_input* (the reader is reminded that "shft" in a variable name indicates the center geometry as discussed in Chapter II) holds the array generated by the function written to model the chosen excitation function.

From *shft_input*, *F_input* is created by shifting (***fftshift***) *shft_input* to a corner geometry and then taking the two dimensional spatial Fourier transform (*fft2*). As explained in Chapter II, the ***fftshift*** operator is necessary before the Fourier operation to maintain the correct phase relationship in the transform operation. With a shift back to the center geometry, the angular spectrum of the source $\tilde{s}(f_x, f_y)$, called *Fshft_input* in the program, is created. The propagation transfer function $\tilde{h}(f_x, f_y, z, t)$ from Eq. (18) must now be loaded so that the angular spectrum–propagation transfer function product $\tilde{s}\tilde{h}$ on the right side of Eq. (21) can be formed. The multiplication

is a repetitive one since *Fshft_input* must form a product with the filter (or appropriate propagation transfer function) from each time slice. This repetitive multiplication is accomplished with a loop.

Looping allows loading of each successive filter $\bar{h}$ to form the product of $\bar{s}$ and $\bar{h}$ (called *Fshft_output*). To get the desired result of Eq. (21), the two-dimensional inverse spatial Fourier transform (*ifft2*) must be taken. Before this can be done, *F_output* is formed by shifting *Fshft_output*. Executing the inverse transform yields *output* which is then shifted to give *shft_output*. The array *shft_output* represents the output of the time slice that the loop is currently computing; *shft_output* does not depict the acoustic potential (or propagation pattern) through time, it only depicts the acoustic potential at a specific time.

To produce the desired output, the center row (row *N0*) of *shft_output* is taken and placed in the matrix *output_plot* as the m^{th} column (m is the loop counter which relates directly to the time slice number, i.e., when $m = 4$, the computations for time slice four are performed). This results in a matrix whose size is *N*x*M* when *Step* zero-rows are added that preceed the fourth time slice's *N0* row. The output examples in Appendices G and H are graphical interpretations of *output_plot*. Results generated in this manner are given in Chapter IV for all of the excitation functions. More examples of *output_plot* generated by the **Gaussian** and **Bessel** excitations are given in Appendices G and H, respectively. A detailed explanation of AC_PROP is provided in Appendix D and the source code in Appendix E.

C. PROGRAM SUMMARY

The previous two sections gave an overview of the two program modules, AC_FIL and AC_PROP, including the code variable names and values assigned. What follows here is a summary of the steps that the program accomplishes. Step one is accomplished by AC_FIL. In this step the $M-Step$ filters to be used by AC_PROP are generated and saved.

AC_PROP generates the user specified excitation function $s(x,y)$. Then the angular spectrum of the source $\tilde{s}(f_x, f_y)$ is computed by taking the 2-D spatial Fourier transform of $s(x,y)$. The product $\tilde{s}\tilde{h}$ is computed for each time slice via a loop. In the loop, the inverse 2-D transform of the $\tilde{s}\tilde{h}$ product forms an output for the specific time slice. The center row (the row that contain singular information) of that output is then placed in successive columns of a new matrix to form the final output, $p(x,0,10,t)$.

In the following chapter, examples of the final outputs are given. The excitations used for verification, as previously related, are the **Table** and **Circle** excitation functions. New values, as explain in this chapter, were then used for the simulation of the **Gaussian** and **Bessel** excitations.

IV. NUMERICAL SIMULATION

In the previous chapter, a functional explanation of the two program modules was given including values assigned. The first section reiterates the defining parameters, gives a brief explanation of each, and gives the parameter's units. In the following section, the defining parameters are given the values used to verify correct operation of the program. The last section presents results for the Gaussian and the Bessel excitation functions.

A. DEFINING PARAMETERS

A defining parameter is a parameter that delineates an aspect of the basic setup upon which all the remaining parameters or variables depend. In the work of this thesis there are two sets of defining parameters — those for the filter generation and those for generation of the excitation functions. The filter parameters are found at the beginning of AC_FIL. AC_PROP queries the user for the excitation function parameters.

The defining parameters found in AC_FIL include N, M, *Step*, c, z, *time_max*, and *rho_max*. The first parameter N sets the dimensions of the base array giving the number of sample points. The dimensions of the base array are, therefore, NxN where N is required to be a power of two. Making N a power of two allows MATLAB to use a high-speed radix-2 fast Fourier transform algorithm [Ref. 12] to

compute the spatial transforms. The next parameter M is the number of time samples; this means there are M time slices. Of these M time slices, $M - Step$ require filters to be computed; the parameter *Step* is the number of leading-zero rows in the NxM output array; as explained in the preceding chapter, this simulates the Heaviside step function. The parameters N, M, and *Step* are unitless and are stored by AC_FIL in a file for use by AC_PROP.

The remaining defining parameters of AC_FIL have units and are used only in the computations of the filters. The acoustic velocity in the medium, free-space in this case, is denoted by the parameter c having the units meters per second. The source-to-reception point distance has the designation z in units of meters. The maximum time of propagation *time_max* has units of seconds. The spatial-frequency radius of the filters *rho_max* (or *rho*) has units of inverse length (i.e., m^{-1}, cm^{-1}, etc.). The unit of length depends on the area to be represented by the base array. These four parameters relate directly to Eq. (18) and are the parameters that dictate the diffraction properties of the filters. Table I summarizes the defining parameters found in AC_FIL.

Another important set of defining parameters is the set that defines the user chosen input. These input defining parameters are entered by the user when requested by AC_PROP. Once the input function is chosen, the diameter of the truncation circle d is input. (The width of the table w (vice d) is input in the case of the **Table** excitation.) The parameters d (or w) are expressed as the number of points, out of N total points, that define the diameter (or width) of the function.

TABLE I. SUMMARY OF THE DEFINING PARAMETERS IN AC_FIL.

DEFINING PARAMETERS	
N	Size of square base array
M	Number of time slices
Step	Number of leading zero-rows
c	Acoustic velocity in media (m/s)
z	Distance, source-to-receiver (m)
time_max	Maximum time of propagation (s)
rho_max	Spatial radius of the filters (length^{-1})

To transition from a diameter in terms of a number of points to an actual metric value, two equations were needed. The equations are

$$\Delta x = \frac{1}{2 * \rho_{max}} \tag{22}$$

and

$$d = k * \Delta x. \tag{23}$$

In Eqs. (22) and (23) Δx is the length of a segment. In Eq. (22) ρ_{max} is the maximum spatial radius. In Eq. (23) k is the number of segments and d is the diameter (or width for the **Table** function). To determine the metric diameter Eq. (22) was used to solve for Δx by setting ρ_{max} to 200 m^{-1}. This value resulted in a Δx of 2.5x10^{-3} m or 2.5 mm.

If the **Gaussian** excitation is selected, the standard deviation σ is input upon request. In a **Bessel** excitation selection the scaling factor a is input when requested. Table II gives a summary of the defining parameters used in AC_PROP.

TABLE II. SUMMARY OF THE DEFINING PARAMETERS USED IN AC_PROP.

DEFINING PARAMETERS	
N	Size of square base array
M	Number of time slices
Step	Number of leading zero-rows
d	Diameter of excitation function (number of sample points)
w	Width of Table excitation function (number of sample points)
σ	**Gaussian** standard deviation
a	**Bessel** scaling factor

B. PROGRAM VERIFICATION

In verifying the program output, two excitation function were used, the **Table** and the **Circle**. The outputs generated by the program from these excitations were compared to the results found in the literature for validation. After a general explanation about the generation, formatting, and titling of the outputs, the two excitation functions are presented. The **Table**, the first verification function, is then discussed and a table of defining parameters is given. The second verification

function, the **Circle**, follows with a similar discussion and table of defining parameters.

1. Results Format

The graphical outputs for the two excitation functions used for verification, the **Table** and the **Circle**, were generated, formatted, and titled in the same manner. The outputs are for a source-to-receiver distance of z = 10 cm. Each shows data for 64 time slices including 3 leading all-zero columns which simulate the step function. Since the outputs were formed by taking the center row of acoustic potential information $p(x,0,10,t)$ for each time slice (placed in the final output matrix as a column), only $1/N$ of the output data is actually presented. This is done so that the three dimensional output images display the magnitude at the receiver plane as a function of the propagation time, up to *time_max*, and radial distance from the center row, about which the outputs are symmetrical. Along with the 3-D image, a side view is provided.

The side views are a plot of magnitude ($|p(x,0,10,t)|$) versus propagation time for various values of x with the same scaling as in the 3-D image. Since the data is symmetric about the radial center, the plot is from the radial center out to and including the first all-zero row. Of course the radial distance to the first all-zero row depends on the maximum time of propagation. The side views, therefore, give numerical information that is more easily extracted for quantitative analysis, as well as other features that are not easily seen in the 3-D images. The spatial excitation function $s(x,y)$ are also given with those of the outputs.

The titling of the images for the **Table** and the **Circle** follows the same general format, *{excitation type}_{diameter/width}* for an input and *{excitation type}O{diameter/width}* for an output. In the case of the side views, the prefix *sv* is added to the output title. Examples of the titling are **T_11**, a **Table** input excitation function of width w = 11 samples; **TO11**, the 3-D output of the **Table** excitation with width 11 samples; and **svTO11**, the side view of the output for the **Table** excitation with width 11 samples.

2. Table Impulse Excitation

The first excitation function to be run by the program was the **Table**. There were several reasons to use the **Table** as the first input; the table function is an easy function to implement and the results could be readily compared to results found in the literature [Refs. 7, 9, 10, and 11]. A list of the defining parameters used is provided in Table III.

The values of N, M, z, *time_max*, *rho_max*, and w chosen in Table III parallel those found in the literature used for validation. The acoustic velocity c of 1500 m/s is the velocity in water. To simulate the step function a number *Step* of leading zero-rows was incorporated and arbitrarily set to three. The results were comparable to those in the literature and are presented (with the input functions) in Figs. 7 and 8.

The input images of Figs. 7 and 8 titled T_11 and T_15 show an input amplitude of one. Here the widths w = 11 and w = 15 samples translate into metric

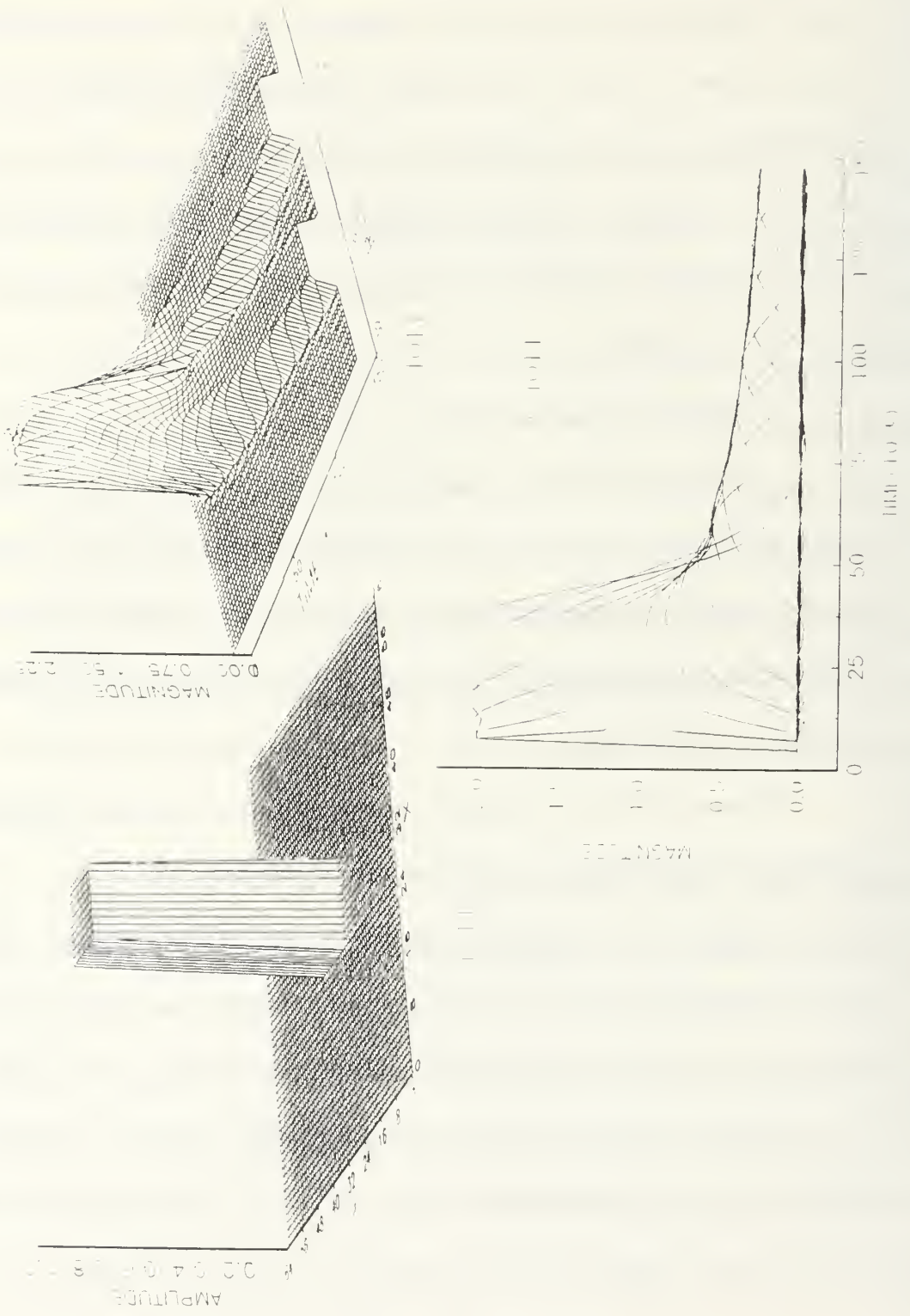

Figure 7. **Table** input and output with output side view for $w = 11$.

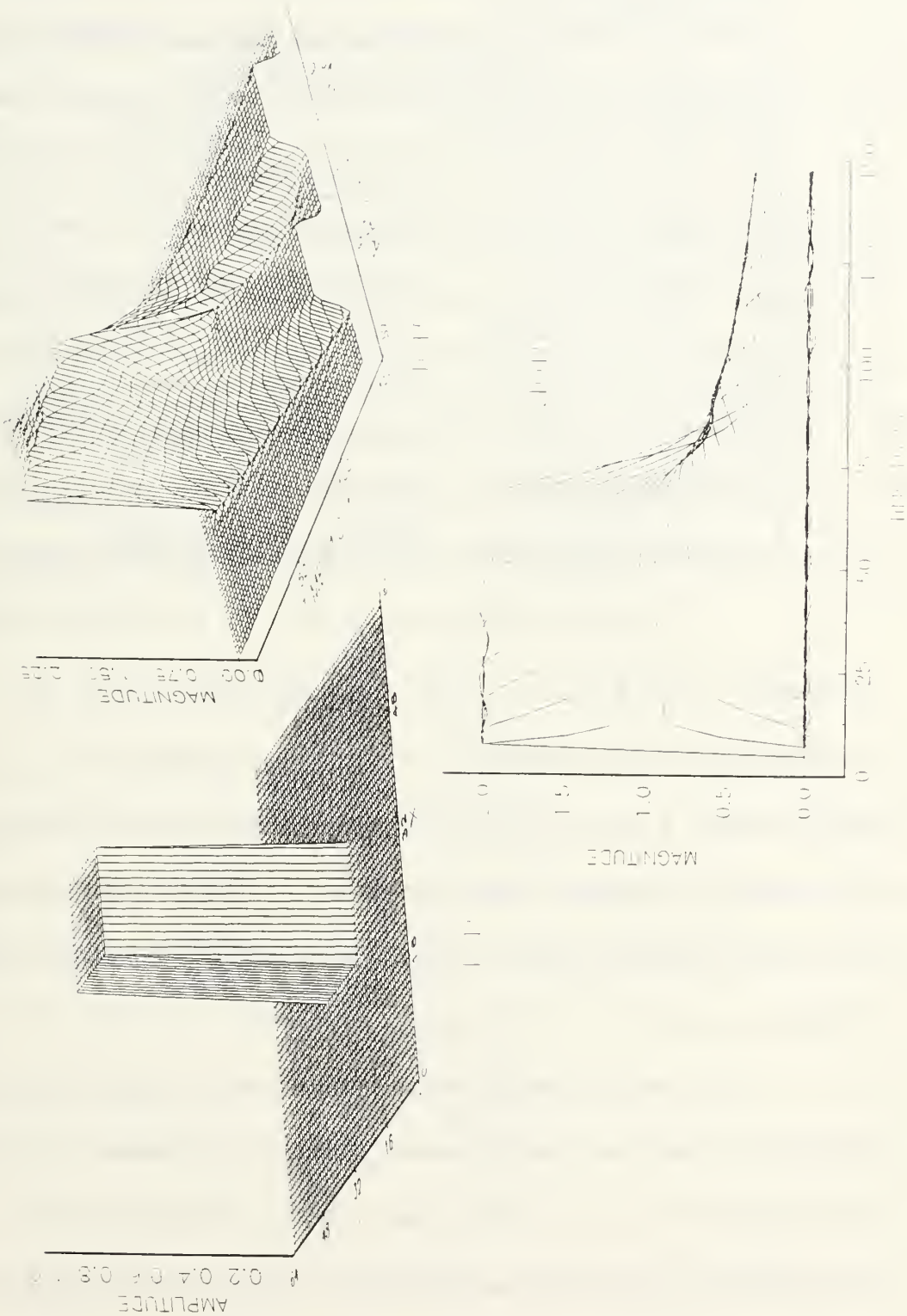

Figure 8. **Table** input and output with output side view for $w = 15$.

TABLE III. DEFINING PARAMETERS FOR THE TABLE USED FOR VERIFICATION.

DEFINING PARAMETERS		
NAME	VALUE	SUMMARY
N	64	Size of square base array
M	64	Number of time slices
Step	3	Number of leading zero-rows
c	1500	Acoustic velocity in media (m/s)
z	0.1	Distance, source-to-receiver (m)
time_max	1.5×10^{-4}	Maximum time of propagation (s)
rho_max	200	Spatial radius of the filters (length^{-1})
w	11, 15	Widths of Table (samples)

widths of $w = 2.5$ cm and $w = 3.5$ cm, respectively. In the $w = 11$ case, $k = 10$ was the input in Eq. (23) and, in the $w = 15$ case, the input was $k = 2.2$ samples. Note that the X and Y axes of the input images range from 0 to 64, delineating a 64x64 base array. The two output images are titled TO11 and TO15 with their respective side views titled svTO11 and svTO15. These output images show several interesting features.

The outputs present the magnitude of the acoustic potential at an observation point 10 cm from the source, $p(x,0,10,t)$. A diffraction duration from the initial impulse ($t = 0$) to $t = time\_max$ (150 μs) is represented as a function of radial distance; i.e., $p(x,0,10,0)$ to $p(x,0,10,150)$ is represented. This gives the 3-D view of the general diffraction through time. At $t = z/c$, after the leading zero-rows, it is

observed that the output is a scaled replica of the input. As time passes, the potential is a combination of waves from various points of the source. The development of "tails," explained in terms of edge waves [Ref. 7], can be seen in the 3-D images. The magnitude of the "tails" approaching zero as time becomes very large is shown in the side views. Also in the side views, the step function is easily identified since the potential remains zero until the fourth time increment (where $t = z/c$) at which point the magnitude "steps" up to a value of two; the factor of two is a result of the multiplicative constant in Eq. (18). Also of note are the overshoots having a maximum magnitude of 2.11 (both cases) and the undershoots (hard to see in these two cases); these are due to the additive nature of the interference patterns of the waves originating on the edges of the discontinuous source.

3. Circle Impulse Excitation

The defining parameters for the **Circle** excitation are the same as those introduced in Table III with the exception that only the $d = 15$ case is presented. A diameter $d = 15$ samples translates into a metric diameter of $d = 3.5$ cm. Again the results are comparable to those found in the literature [Refs. 9 and 11]. Figure 9 gives the input function and the ensuing outputs. As in the **Table** case the base array is a 64x64 matrix with the propagation pattern formed by successive $p(x,0,10,t)$ time slices. The results in Fig. 9 for the **Circle** excitation are much the same as those for the **Table** in Fig. 8. The "tails," however, shown in the 3-D image of Fig. 9 are rounded instead of cornered as in the **Table** output. Though the **Table** output holds its magnitude for a longer time, the maximum for the **Circle** output is greater at 2.21.

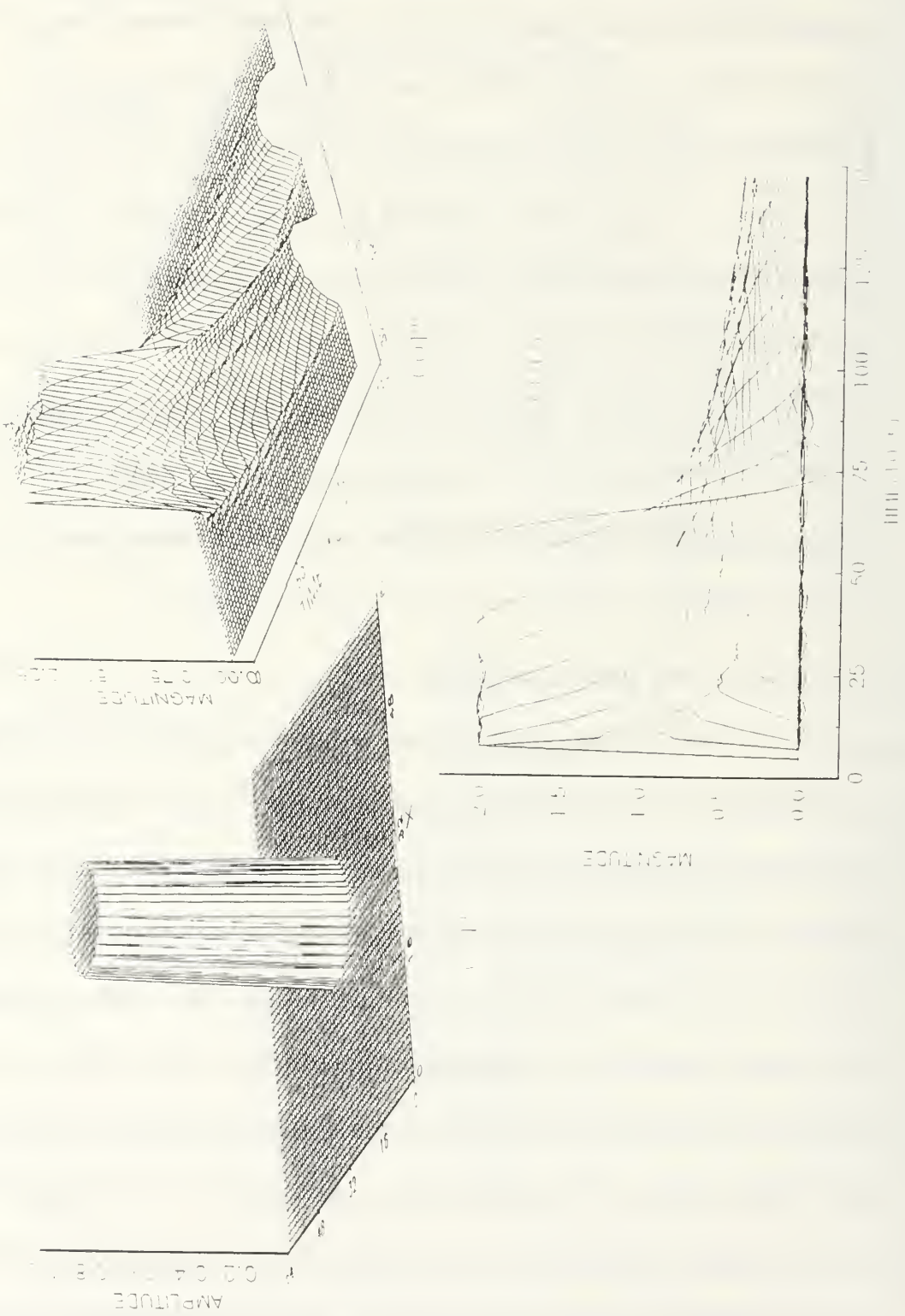

Figure 9. **Circle** input excitation and output with side view for $d = 15$.

Also the drop off from the maximum is steeper for the **Circle**. The greater maximum and steeper drop off are due to the equal distance of all edge points from the center. This same geometric influence also accounts for the **Circle** holding the input value for a shorter duration. Just as the interference patterns added to a maximum greater than the input, the interference patterns also combine to give a more negative minimum. This lower minimum is easily seen in the side view image in Fig. 9. The negative undershoot was present for the **Table**; however, it has a magnitude five times greater for the **Circle**.

C. OTHER INPUT EXCITATIONS

Having verified the operation of the program to a high degree of confidence, the defining parameters were changed and other excitation functions evaluated. The first change was to increase N by a factor of two from 64 to 128. Previously, software and computer hardware limited the size of the base array to a 64x64. The increase to $N = 128$ translated to increased sampling and spatial resolution. Since the size of the base array was increased, the base diameter d of each excitation function must be increased (double the number of segments) in order to model a source with the same metric size as in the 64x64 case. This simply translates to increased spatial sampling. A diameter of $d = 51$ samples (a metric diameter of $d = 12.5$ cm from Eq. (23) when $k = 50$ is the input) gave the best graphic results and was therefore used as the base diameter. The changes in N and d motivated a change in the propagation time as well.

Increasing the size of the base array allowed an increase in the widths of the excitation functions. This necessitated a longer propagation time so that the full effects of propagation could develop. As a result, the maximum time of propagation *time_max* was increased to 375 μs. This value of *time_max* allowed the "tails" to form and present a trend.

Once the values of N, d, and *time_max* were set, the program was ready for the other excitation functions. The first to be presented is a circularly truncated **Gaussian** distribution function. Following the **Gaussian**, a circularly truncated **Bessel** profiled function is examined. These two excitation function outputs are generated and formatted the same. The titles have a similar structure.

As with the **Table** and **Circle**, the **Gaussian** and **Bessel** outputs are generated by a successive compilation of the $p(x,0,10,t)$ vectors. This means that all the x-values of acoustic potential at $y = 0$ and $z = 10$ cm at time t were placed in successive rows of the output matrices. In these cases, however, time started at $t = z/c$ and ran until *time_max* of 375 μs in 61 time slices. The result was a 64x128 matrix of acoustic potential as a function of propagation time and radial distance once the simulated step function was added. As with the previous excitation function the input images represent the spatially arbitrary (but assumed known) source $s(x,y)$ that is impulsive in time.

Titling of the input and output images is similar to that used for the **Table** and **Circle**; *(excitation type)_(parametric information)* for an input and *(excitation type)O(parametric information)* for an output. The *(parametric information)* has the form *(standard deviation)x(base diameter)* for the **Gaussian** and *(scaling factor times 10^4)* for the **Bessel**. Additionally, the side views have the prefix *sv* appended. Input examples are G_5x51 for a **Gaussian** with a standard deviation $\sigma = 5$ and B_2500 for a **Bessel** with scaling factor $a = 1/4$. Both having a base diameter $d = 51$.

1. Gaussian Distributed Excitation

Though the **Gaussian** has been investigated before [Refs. 7, 8, 9, and 10], it has not been studied as a 128x128 array. For convenience, the defining parameters are listed in Table IV. Figure 10 is the input and resulting output where the input image is titled G_5x51 and the output image is titled GO5x51.

The **Gaussian** excitation function has been normalized by the maximum value of the computed Gaussian (see the **Gaussian** source code titled CRCGAUS.M in Appendix F). This normalization is shown in the input image of Fig. 10 by the maximum amplitude of one. Displayed as an *N*x*N* array, this input image has a standard deviation of $\sigma = 5$ and a 1/e point of 10.17 samples from the center, having metric equivalent of $r = 2.54$ cm. The diffraction of this input is shown in the output images, GO5x51 and svGO5x51, of Fig. 10.

The 3-D image shows a diffraction pattern that is well established by time $t = time\_max$, forming two spreading "tails." The "tails," as well as the rest of the

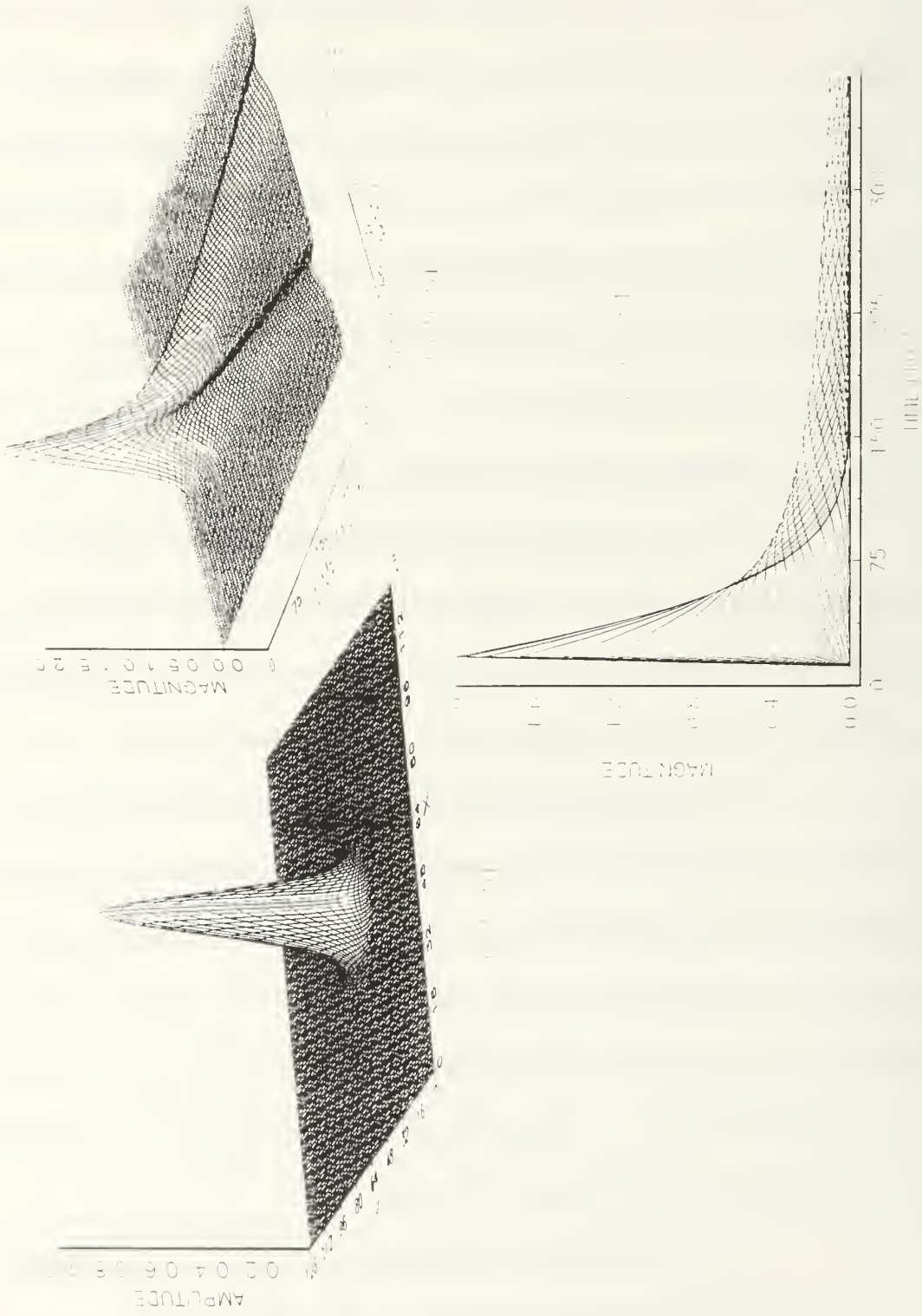

Figure 10. **Gaussian** distributed input and output with side view for $\sigma = 5$.

TABLE IV. EXAMPLE DEFINING PARAMETERS FOR A **GAUSSIAN** EXCITATION.

DEFINING PARAMETERS		
NAME	VALUE	SUMMARY
N	128	Size of square base array
M	64	Number of time slices
Step	3	Number of leading zero-rows
c	1500	Acoustic velocity in media (m/s)
z	0.1	Distance, source-to-receiver (m)
time_max	375×10^{-6}	Maximum time of propagation (s)
rho_max	200	Spatial radius of filters ($length^{-1}$)
d	51	Diameter of excitation function (samples)
σ	5	Gaussian standard deviation

diffraction pattern, are smoothly rounded. This rounding is the result of the continuity of the Gaussian distribution. A discontinuity, as in the previous two excitation shapes, would result in a characteristic over and undershoot of the maximum and minimum inputs. Again, the results for this **Gaussian** excitation conformed to those found in the literature [Refs. 7-10]. The **Bessel** excitation was then run and the results compared to those of the **Gaussian.**

2. Bessel-Profiled Excitation

Results for the **Bessel** excitation produced by CRCBES.M (Appendix F) were generated, as previously discussed, for the set of defining parameters listed in Table V. The resulting input and output are shown in Fig. 11. In Fig. 11 the input and output images are respectively titled B_2500 and BO2500. Since the output is formed by taking successive $p(x,0,10,t)$ vectors, three peaks appear in the 3-D image. These three peaks correspond to the center peak and the points on the two crests directly adjacent to the center. As a result of having three peaks, three "tails" are present. The "tails," however, are smooth because a Bessel function is a continuous function. Also worth noting in the 3-D image is the simulated step function, more visible here due to the oscillations of a Bessel. The output side view shows the positive to negative oscillations of a Bessel function.

Comparing the **Gaussian** and **Bessel** outputs, it is seen that the **Gaussian's** magnitude retention is slightly longer than that of the **Bessel**. This is due to the more gradual decrease in magnitude vice the steeper decrease required of the **Bessel** so that it can become negative. Still no hard conclusions can be drawn without further analysis.

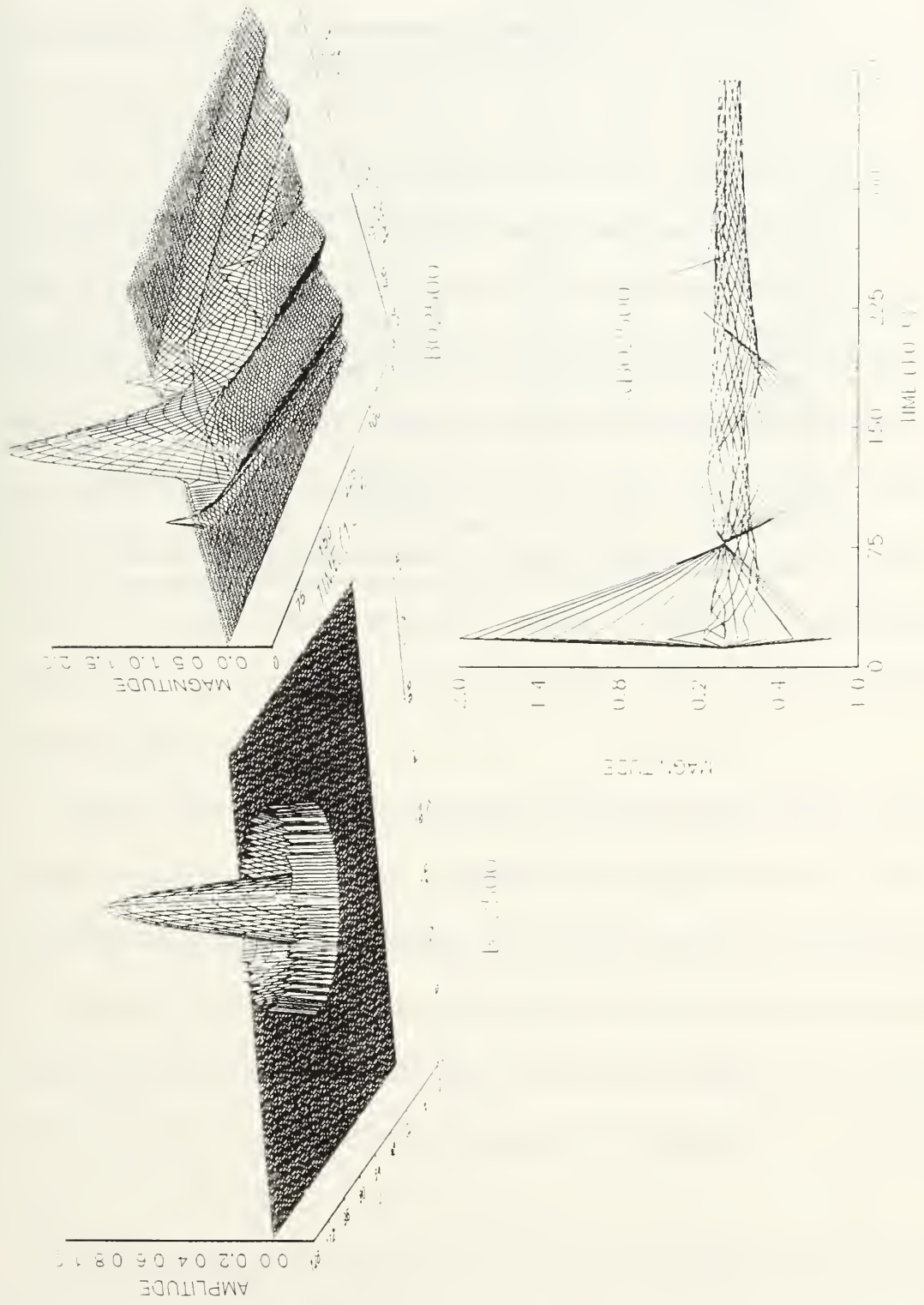

Figure 11. **Bessel** profiled input and output with side view for $a = 1/4$.

TABLE V. EXAMPLE DEFINING PARAMETERS FOR A BESSEL EXCITATION.

DEFINING PARAMETERS		
N	128	Size of square base array
M	64	Number of time slices
Step	*z*	Number of leading zero-rows
c	1500	Acoustic velocity in media (m/s)
z	0.1	Distance, source-to-receiver (m)
time_max	375×10^{-6}	Maximum time of propagation (s)
rho_max	200	Spatial radius of filters (length^{-1})
d	51	Diameter of excitation function (samples)
a	1/4	Bessel scaling factor

V. SUMMARY

This thesis presented a MATLAB implementation of a Fourier approach to ultrasonic wave propagation. A mathematical development using linear systems that found the acoustic potential from an arbitrary spatial and temporal source was presented. In the mathematical development, it was shown that the Green's function solving the appropriate wave equation and satisfying the boundary conditions is the total impulse response of the system. Through double and triple convolutions, the acoustic potential could be found for any source separable in time and space. Use of the 2-D spatial Fourier transform, however, translated the convolution to multiplication in the spatial frequency domain. This made the MATLAB implementation easier.

After an overview of MATLAB and the graphics program AXUM, a functional description of the program was furnished. The program modules AC_FIL and AC_PROP both made use of symmetry. AC_FIL generated the time-varying filters, the most time consuming process, while AC_PROP accomplished the remaining computations making use of MATLAB's "fft2" function. Details of both modules as well as the source code have been included in the Appendices for the interested reader.

Several examples were delineated in the body of this thesis. First, the **Table** and

the **Circle** were presented as the program verification excitations; the results conformed to those found in the literature. Then two newer excitation functions, the **Gaussian** and the **Bessel**, were presented. More examples of the **Gaussian** and the **Bessel** have been included in the Appendices.

The underlying result was an accurate and efficient computer implementation of the linear systems approach to ultrasonic wave propagation. The efficiency was derived from the modularization of the program so that consecutive runs could be made without recomputing the most time consuming portion, the filters. Also, the use of MATLAB's "fft2" function bypassed tedious and time-inefficient convolution integrals. Finally, both modules made use of symmetry by computing only one quadrant of data which was then manipulated into the remaining quadrants. An advantage to using MATLAB was the ease of expansion that could be accomplished with the program.

The work of this thesis concentrated on a source with rigidly baffled boundary conditions and a lossless media. Cases that include free space and resiliently baffled boundary conditions as well as lossy media, linear and quadratic lossy media, could be incorporated. A few facts worth noting here are that the free space and resilient baffle boundary conditions can be expressed in terms of the rigid baffle case [Ref. 8] and that the lossy media and lossless media transfer functions are interdependent [Ref. 10]. Furthermore, new excitation functions such as a phased array or a focused source could also be incorporated. Improvements are needed in the area of analysis such as the **Gaussian** versus **Bessel** propagation comparison and extending the

technique to sources that are not time and space separable such as new non-diffracting waves [Ref. 14].

APPENDIX A. DETAILED EXPLANATION OF AC_FIL.M

The following explanation addressing the first program module AC_FIL is intended to supplement the functional explanation found in the body of this thesis and the comments in the source code found in Appendix B. Prior knowledge of MATLAB is assumed, as is familiarity with information previously presented. In this explanation, MATLAB commands will be lower-case **bold** and variables will be as they appear in the code and in *italics*. The reader is referred to Ref. 12 for more details on the MATLAB commands contained in this appendix.

The primary task of AC_FIL is the generation of the time-varying, Bessel profiled, spatial filters. AC_FIL opens up with some preliminary housekeeping by clearing the RAM and deleting all the **.mat** files from previous runs. Having cleaned up, the next chore is to set the values of the defining parameters (*N*, *M*, *Step*, *c*, *z*, *time_max*, and *rho_max*) and compute *N0* (a function of *N*). Having set the defining parameters, the four matrices and two vectors used throughout the program are initialized with the **zeros** command. This initialization saves processing time [Ref. 12]. The next duty is the formation of the *time* vector using the command **linspace**. The **linspace** command generates a vector whose $M - Step$ points are linearly spaced from *z/c* to *time_max*, inclusive. The ensuing task makes use of the **save** command to create a file named AC_FIL.MAT in which the variables *N*, *N0*, *M*, and *Step* are

saved for use in AC_PROP. The next several procedures compute the non-time varying portion of the argument for the Bessel function.

The first *N0−1* terms of the vector *rhom* are given the values from 0 to *rho_max* with the **linspace** command. Since the larger of the quadrants has *N0* columns, *rhom* is modified by stipulating as the *N0* term the sum of the second term and the *N0−1* term. Then two arrays *rhox* and *rhoy* are created from *rhom* by the **meshdom** command. The two arrays are *N0xN0* arrays where each row has the corresponding value from the *rhom* vector (i.e., row one has the value from column one, row *N0* has the value from column *N0*, etc.). The radial distance ρ, called *rho* in AC_FIL, is the square root of the sum of the squares of *rhox* and *rhoy*. The elements of *rho* are computed on an element-by-element basis; this is accomplished by the dots in the command line ***rho* = sqrt(*rhox*.^2 + *rhoy*.^2)** and results in an *N0xN0 rho* matrix. That completes the computation of the non-time varying portion of the Bessel argument.

The remainder of the program is a loop which computes the filter for each time slice. A loop counter *m* running from 1 to *M−Step* is utilized for the 61 time slices with the number of the current time slice displayed as a result of the **fprintf** command. The next two chores compute an *N0xN0* Bessel array named *temp* whose maximum value is *temp*(33,1) ((row,column) format) giving *temp* a quadrant I orientation.

First, the Bessel function argument *arg* is computed as the product of *rho* and **sqrt((*c*^2)*(*time*(*m*)^2)−(*z*^2))**. Subsequently, *temp* is calculated by the **besseln**

function with arguments 0 and *arg*; here, 0 is the integer order of the Bessel function. The filter variable *PROP* is then generated from *temp*.

Each *PROP* (or filter) is an *NxN* array formed by setting rows 1 to *N0* and columns *N0* to *N* of *PROP* equal to rows 1 to *N0* and columns 1 to *N0 – 1* of *temp*. Quadrant II of *PROP* is formed by the command **fliplr** which flips *temp* about column one. A similar command **flipud** is used to flip rows 2 through *N0* of *PROP* about row *N0* completing the formation of the entire *PROP* array. This construction can be viewed by removing the optional comment indicators *%*%* found directly before the **mesh** commands. The final task is saving the filter *PROP* for use by AC_PROP.

Saving *PROP* is a two step sequence. Step one of the sequence appends the time slice number *m* to each *PROP*. This is accomplished by setting the variable *vname* equal to the string PROP_0*m* (for $m < 10$) or PROP_*m* (for $m > 10$) in which the command **int2str** replaces *m* with its numerical value. The **eval** command then assigns the values found in *PROP* to the string found in *vname*; this effectively renames each filter *PROP*_{time slice number}. To save each filter, the **eval** command is again employed. The two command lines

eval(['save ac',int2str(*N*),'x0',int2str(*m*),' ',*vname*]);
%%* **eval(['save e:\AC_OUT\ac',int2str(*N*),'x0',int2str(*m*),'.dat', *vname*,'/ascii']);**

both save each filter in a file named AC*N*x0*m* where **int2str** replaces the variables with the current values. The second **eval** command, however, saves the file with a .DAT suffix in ASCII format to the E disk drive in the directory AC_OUT. The

above command line examples are for the $m < 10$ filters; there are similar command lines for the $m > 10$ filters. The final step in the loop is to ready *PROP* and *vname* for the next pass by clearing *PROP* and *vname*.

APPENDIX B. SOURCE CODE FOR AC_FIL.M

The following is the source code used to generate the time-varying filters as discussed in Chapters II and III. AC_FIL.M was written in block format with each block headed by a descriptive comment to explain the block's function. The code includes many optional instructions indicated by the leading %*% symbol. Deleting this symbol will enact that line of code on succeeding program runs. This, of course, varies the output; however, the outputs necessary to a successful run of AC_PROP.M are the file AC_FIL.MAT and the files AC*Nxm*.MAT (*m* is an index number from 01 to 61). AC_FIL.MAT contains variables needed in AC_PROP.M. The AC*Nxm* files contain the filters (in a variable named PROP_*m*) that correspond to the time slice index *m*.

AC_FIL.M SOURCE CODE

```
%%%%%          **** AC_FIL.M ****
%% This program generates an Acoustic Propagation Transfer
%% Function, a time varying spatial filter, for use in
%% AC_PROP.M to simulate acoustic wave diffraction.
%%        William H. Reid          December 1992

 clear;                          % Remove all variables from RAM.
 !del ac*x*.mat                  % Remove files from a previous run.

 N = 128;                        % Size of square base array.
 M = 64;                         % Number of time slices.
 N0 = (N/2)+1;                   % Defines center of square base array.
 Step = 3;                       % Number of leading zero time slices,
                                 %  simulates the step function.
 c = 1500;                       % Velocity of the acoustic wave, (m/s).
 z = 0.1;                        % Distance to the observation plane, (m).
 time_max = 3.75e-4;             % Maximum time of propagation,
```

```
                                % time at the final time slice, (sec).
rho_max = 200;                      % Spatial radius of the filter.
                                % [sqrt(rhox^2 + rhoy^2].

%% Matrices initialization to reduce processing time.
  PROP = zeros(N);   temp = zeros(N0);
  arg = zeros(N0);   rhom = zeros(N0,1);
  rho = zeros(N0);   time = zeros(M-Step,1);

%% Generate M-Step time slices between z/c and time_max.
  time = linspace(z/c,time_max,M-Step);

%% Save those variables necessary for AC_PROP.M.
  save AC_FIL N N0 M Step;

%% Generate N0-1 values of "rhom" from 0 to rho_max.
  rhom = linspace(0,rho_max,N0-1);

%% Add additional increment to rhom to compensate for off center
%% orientation of the final N x N matrix.
  rhom = [rhom (rhom(N0-1)+rhom(2))];

%% Create two N0 x N0 arrays of rho_max values for function evaluation.
  [rhox,rhoy] = meshdom(rhom,rhom);

%% Calculate "rho", an N0 x N0 matrix of radial distances for use in
%% the argument to the Bessel function within the loop.
  rho= sqrt(rhox.^2 + rhoy.^2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% START LOOP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Generate M-Step filter matrices, the filter at each time slice.
  for m = 1:(M - Step)
    fprintf('%3.0f',m);      % Display m value for user progress report.

%% Create an N0 x N0 array of argument values for the Bessel function.
    arg = row * sqrt( (c^2)*(time(m)^2)-(z^2) );

%% Evaluate the zero order Bessel for each argument value,
%% creates an N0 x N0 array called "temp".
    temp = 2*besseln(0,arg);

%% Form each N x N filter matrix called "PROP" from "temp".
%% (The optional mesh commands are for viewing the construction steps.)
    PROP(1:N0,N0:N) = temp(1:N0,1:N0-1);
     %*% mesh(PROP);title ('Quadrant I');pause
    PROP(1:N0,1:N0) = fliplr(temp);
     %*% mesh(PROP);title('Quadrants I & II'); pause
    PROP(N0:N,1:N) =flipud(PROP(2:N0,1:N));
     %*% mesh(PROP);title(['Filter at time slice ',int2str(m)]); pause
```

```
%% Append the time slice number (loop iteration) to the variable
%% name PROP, i.e. PROP_01, PROP_02,... for m < 10 and PROP_10,
%% PROP_11,... for m = 10 to M-Step
    if m < 10,
      vname = ['PROP_0',int2str(m)];
      eval([vname,'= PROP ;']);
    else
      vname = ['PROP_',int2str(m)];
      eval([vname,'= PROP ;']);
    end

%% Save each time slice filter in a file named ac(N)x(m),
%% (i.e. PROP_05 in acl28x05). The files can be saved to another drive
%% path in ascii format for use with other graphics software.
    if m < 10,
      eval(['save ac',int2str(N),'x0',int2str(m),' ',vname  ] );
   %*% eval(['save e:\AC_OUT\ac',int2str(N),'x0',int2str(m),'.dat',
⇒           vname,'/ascii']);
    else
      eval(['save ac',int2str(N),'x',int2str(m),' ',vname  ] );
   %*% eval(['save e:\AC_OUT\ac',int2str(N),'x',int2str(m),'.dat',
⇒           vname,'/ascii']);
    end
      eval(['clear PROP ',vname]);      % Clear PROP for next loop pass.

 end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% END LOOP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

APPENDIX C. EXAMPLES OF THE TIME-VARYING BESSEL FILTERS

This appendix contains examples of the time-varying filters produced by AC_FIL. Only a select few of the total number M (61 in the cases given throughout this thesis) are provided. The first at time z/c is a plane with amplitude two; this is because the input argument to the Bessel function is zero giving an output value of one. This uniform plane produces an output for time slice one that is a scaled replica of the input. The remainder of the filters illustrate the time variance and how the filters collapse inward with time. Each filter is a 128x128 array representation in the spatial frequency domain.

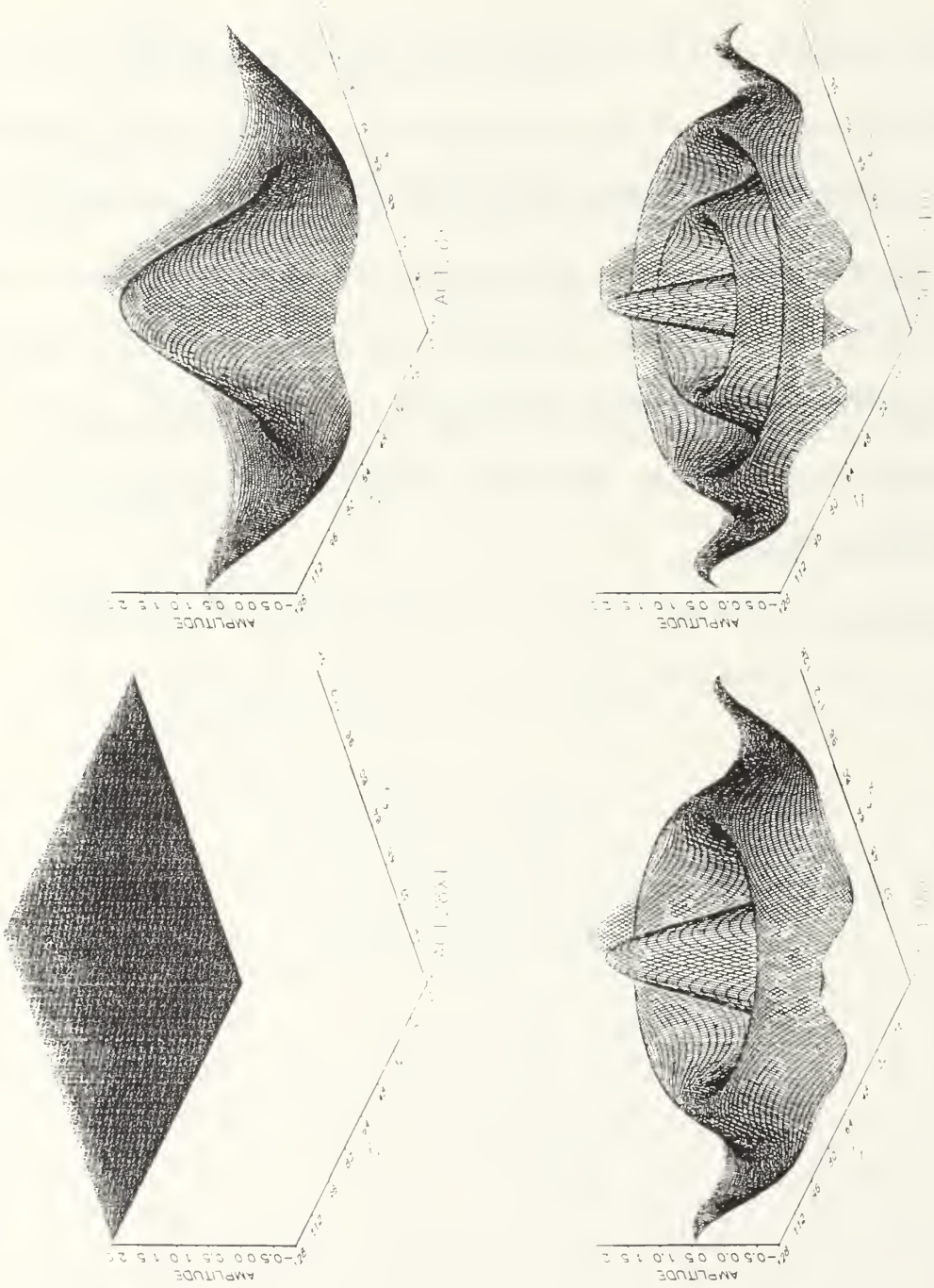

Figure 12. Example filter at time slices 1, 2, 5, and 10.

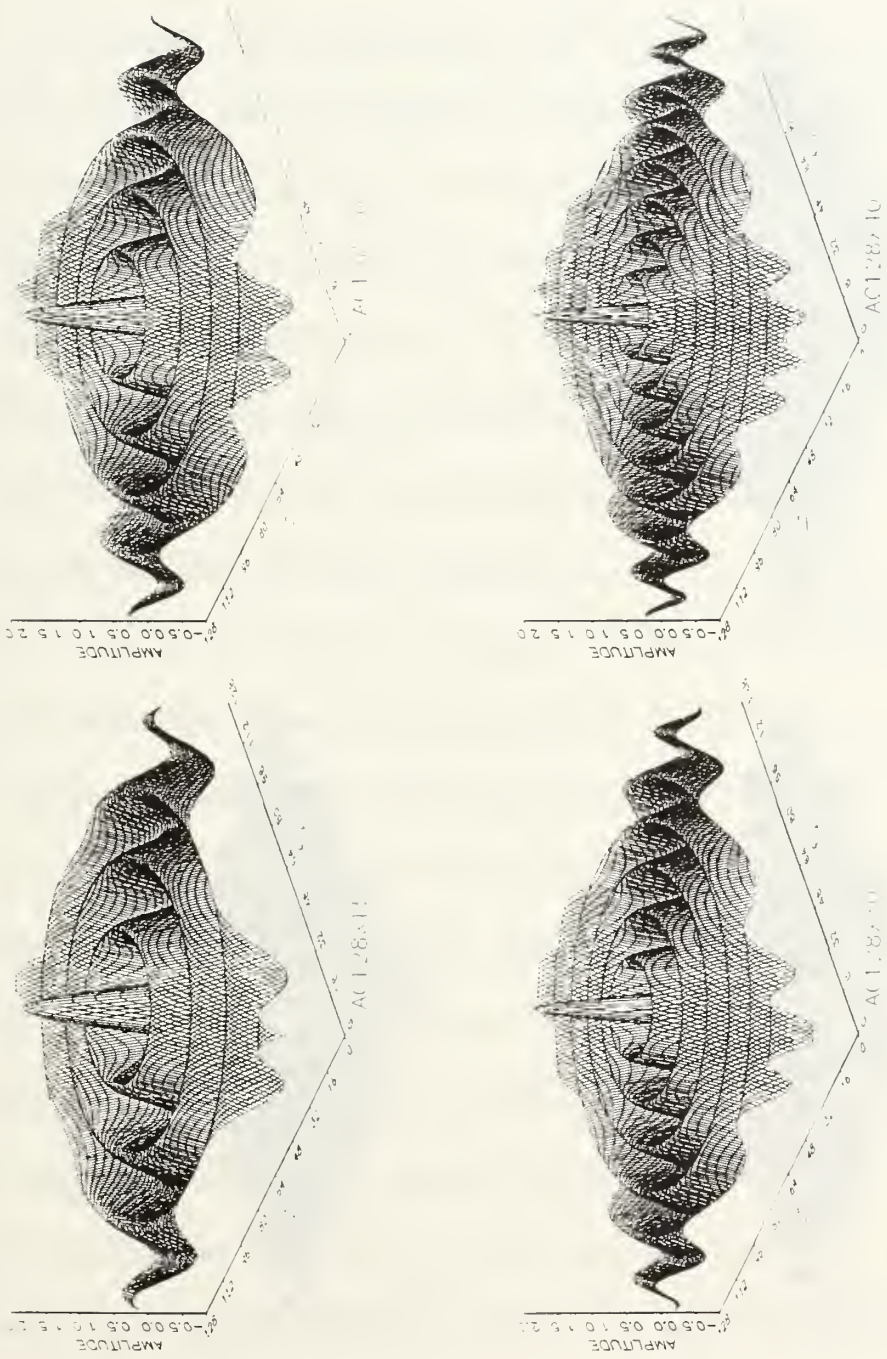

Figure 13. Example filter at time slices 15, 20, 30, and 40.

Figure 14. Example filter at time slices 50, 60, and 61.

APPENDIX D. DETAILED EXPLANATION OF AC_PROP.M

The following explanation addresses the program module AC_PROP. This explanation assumes familiarity with MATLAB and the functional explanation in the body of this thesis. The source code for AC_PROP is given in Appendix E. The MATLAB commands in this explanation are in lower-case **bold** and the variables are in *italics* as they appear in the code. The reader is referred to Ref. 12 for more details on the MATLAB commands contained in this appendix.

AC_PROP starts off with housecleaning by clearing RAM, the screen, and any residual graphics. The next task is to **load** the file AC_FIL.MAT which contains the variables *N*, *N0*, *M*, and *Step*. Then the command **format compact** is executed to save space on the monitor screen for the interactive portion of the program.

The next block of code uses the **disp** command to display information for the user and the **input** command to retrieve the user's input. The user is requested to select one of the four input excitation functions or to strike "enter" to select the default function found in square brackets (i.e., [ ]). After selection of the excitation function, the program then requests values for the appropriate function-defining parameters; again, default values are found in square brackets. A function call is then made to the selected function file and the excitation function is generated and returned to the variable *shft_input* ("shft" in a variable name indicates center

geometry; see Chapter II). Two files called *InFile_Name* and *OutFile_Name* are then created. These files are used for exporting the input function and output data, respectively, to AXUM. The image titles in the figures throughout the thesis and appendices are the names found in *InFile_Name* and *OutFile_Name*. AC_PROP's next chore is saving *InFile_Name* to the indicated directory as a .DAT file in ASCII format and then clearing *InFile_Name* from RAM. The next several blocks of code operate on the chosen input function.

First, the **mesh** command generates a 3-D graphic of *shft_input*. The command **subplot(221)** that precedes **mesh** divides the screen into two sections vertically and horizontally. This forms quadrants with the input graphic being placed in quadrant 1. The **subplot** is executed so that the input and output can be viewed together at the program's end.

An option to generate a MATLAB formatted file of the graphic, called a **meta** file, follows. The **meta** file is used to produce a hard copy of the graphic from MATLAB. Options, indicated with %*%, are included throughout the code. The option to **mesh** for viewing follows each manipulation in the process of formulating the output. Other options include viewing the absolute value of the 2-D Fourier transform of the input, viewing the absolute value of the output, and saving the output of each time slice.

After viewing *shft_input*, *input* is formed by using the **fftshift** command to shift the input to the corner geometry (recall the discussion of Fig. (4)). Shifting of the input prior to taking the 2-D Fourier transform is necessary to maintain the proper

phase relationship in the transform (as discussed in Chapter II). The next procedure is taking the 2-D Fourier transform of *input* to form *F_input*.The **fft2** command effects the transform. Once in the transform domain, the time domain representations *input* and *shft_input* are cleared from RAM. The remaining input process is the **fftshift** of *F_input* back to the center geometry forming *Fshft_input*. RAM is again freed by removal of *F_input*. The **disp** command then informs the user that the program is "performing array multiplication."

The array multiplication is a repetitive operation in that it is performed for each time slice. As with the filter module, a loop with counter *m* is employed. Running from 1 to *M − Step*, the current time slice *m* is displayed utilizing the **fprintf** command. The first operation in the loop is to **load** the AC*N*x*m* file that contains filter *m*. This is accomplished in two steps; first the variable *filename1* is used to assemble the correct file name. The ensuing **eval** command executes the **load** command on the file name contained in *filename1*. Then *filename1* is cleared from RAM. Once the file is loaded into RAM, the program has access to the filter; however, the name of the filter is not known to the program. Again the **eval** command is employed so that *vname1* holds the filter name PROP_*m*. Now the filter input product is computed on an element-by-element basis. The command line ***Fshft_output*** **=** **(*vname1* .* (*Fshft_input*));** produces the transfer domain center geometry output *Fshft_output*.

The output *Fshft_output* is then transformed back to the time domain in a three-step procedure. Steps one and three are shifts using the **fftshift** command at

both ends of the inverse 2-D spatial Fourier transform **ifft2** command. The result is the output *shft_output* for the m^{th} time slice. After clearing RAM, the center row (row *N0*) of *shft_output* is placed in column $m + Step$ of the final output *output_plot*. Before the loop ends, an option to save and export *shft_output* for each time slice has been included. To exercise this or any option, simply remove the %*% at the start of the command line.

When m is equal to $M - Step$, the program exits the loop where the contents of *output_plot* are saved to the appropriate directory in either MATLAB format or ASCII format. The succeeding block of code gives the option of viewing the output in three different views and producing a MATLAB **meta** file. Finally, the absolute value of the output is displayed in quadrant 4 with the input in quadrant 1 via the **subplot(224)** command.

APPENDIX E. SOURCE CODE FOR AC_PROP.M

The following is the source code for AC_PROP.M used to produce various outputs including those discussed in Chapter IV. AC_PROP was written in blocks with each block headed with a descriptive comment to explain the block's function. This also makes it easy to follow the computation from inputs to output. Inputs to AC_PROP are imported from AC_FIL in the file AC_FIL.MAT and the files AC*Nxm*.MAT (*m* is an index number from 01 to 61). The program solicits user input to determine the input excitation. AC_PROP then computes the output.

The format of the output can be changed by the user. Before running the program, the user can remove the optional comment markers indicated with %*% to produce and/or view the output in the desired format. This allows the data to be saved and exported in ASCII format for use with other graphics programs (such as AXUM, see Chapter II).

AC_PROP.M SOURCE CODE

```
%%%%%                  **** AC_PROP.M ****
%% This program performs transient-wave acoustic propagation
%% simulations. It uses the time varying spatial filters called PROP_m
%% to compute the Acoustic Propagation Transfer Function. The PROP_m
%% files are generated by AC_FIL.M and are titled "acNxm.mat" where
%% "N" and "m" are numbers.
%%       William H. Reid                  December 1992

%*% !del ABS_OUT.met    % Remove output from last run.

 clear; clg; clc;
```

```
%% Load the parameters generated by AC_FIL.M.
 load AC_FIL.MAT

 format compact          % Set compact format for screen display.

%% Generate the INPUT function from user interface.
 N                       % Display the size of the square base array.
 disp('N is the width of the base.');  disp('   ');
 disp('Please select the excitation function:');
 disp(' ===>  C - Circle  T - Table  G - Gaussian  B - Bessel <===');
 disp('  ');
 disp('Please, capital letters only! Strike "enter" after selection.');
 disp(' Default values are in [ ].');

 input_func = input('Please enter an excitation function letter
⇒                   [G]:','s');
     if isempty(input_func)
           input_func = 'G'
     end

 if input_func == 'C',
        d = input('Please enter an ODD diameter, [51], d =  ');
           if isempty(d)
                 d = 51
           end
        shft_input = crcle(d,N);
        InFile_Name = [input_func,'_',int2str(d)];     % Name a file to
                                                % hold the input function.
        OutFile_Name = [input_func,'O',int2str(d)];    % Name a file to
                                                % hold the output.

 elseif input_func == 'T',
        w = input('Please enter an ODD width, [11], w =  ');
           if isempty(w)
              w = 11
           end
        shft_input = table(w,N);
        d = w;
        InFile_Name = [input_func,'_',int2str(d)];
        OutFile_Name = [input_func,'O',int2str(d)];

 elseif input_func == 'G',
        sigma = input('Please enter the standard deviation, [10],
⇒                    sigma =  ');
           if isempty(sigma)
                 sigma = 10
           end
        d = input('Please enter an ODD diameter, [51], d =  ');
           if isempty(d)
                 d = 51
```

```
        end
    shft_input = crcgaus(sigma,d,N);
    InFile_Name = [input_func,'_',int2str(sigma),'x',int2str(d)];
    OutFile_Name = [input_func,'O',int2str(sigma),'x',int2str(d)];

elseif input_func == 'B',
    a = input('Please enter a width scaling factor,[.3125], a =  ');
        if isempty(a)
            a = 0.3125
        end
    d = input('Please enter the ODD diameter, [51], d =  ');
        if isempty(d)
            d = 51
        end
    shft_input = crcbess(a,d,N);
    q = a * 1e4;
    InFile_Name = [input_func,'_',int2str(q)];
    OutFile_Name = [input_func,'O',int2str(q)];

else
    disp('   ');
    disp('Incorrect Excitation Function Selection!');
    error('Restart AC_PROP...to try again.');
end

%% Save the input function placed in "InFile_Name" in ascii format for
%% use with other graphics software.
 eval(['save e:\',InFile_Name,'.dat shft_input /ascii']);

%% Remove "InFile_Name" from memory.
 clear InFile_Name;

%% Display the input function for viewing in a subplot so the output
%% can also be display.
 subplot(221),mesh(shft_input);title('SHFT_INPUT'); pause(2);
 %*% meta InPut          % Optional, used to create a MATLAB hard copy.
 %*% clg

%% Shift input quadrants and take the 2-D FFT to produce F_INPUT.
%% Shifting swaps quadrants I & III and II & IV. This is necessary
%% for taking the 2-D Fourier transform (fft2).
 input = (fftshift(shft_input));
   %*% mesh(input);title('INPUT ')          % Optional for viewing INPUT.
   %*% pause(2);clg
 F_input = fft2(input);
   %*% mesh(F_input);title('F_INPUT ')    % Optional for viewing F_INPUT.
   %*% pause(2);clg
 clear input shft_input;                                    % Free RAM.

%% Shift F_input in preparation of multiplication with PROP_m.
```

```
 Fshft_input = fftshift(F_input);
   %*% mesh(Fshft_input);title('FSHFT_INPUT ')    % Optional for viewing
   %*% pause(2);clg                                      % Fshft_input.
 clear F_input;                                          % Free RAM.

%% Plot the absolute value of the fft2 of "shft_input." This is opt onal
%% and is for viewing only.
 %*% mesh(abs(Fshft_input)); title('ABS(FSHIFT_INPUT)')
 %*% pause(2);clg

%% Element by element arrray multiplication of the transfer function
%% filter in "PROP_m" and  "Fshft_input."
 disp('Performing array multiplication....');
%%%%%%%%%%%%%%%%%%%%%%%%%%%% START LOOP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 for m = 1:M-Step
    fprintf('%2.0f,',m)      % Display m value for user progress report.

%% Give the variable "filenamel" the name of the file containing
%% "PROP_m" and then load that file.
    if m < 10,
      filenamel = ['ac',int2str(N),'x0',int2str(m)];
    else
      filenamel = ['ac',int2str(N),'x',int2str(m)];
    end
    eval(['load ',filenamel] );
    clear filenamel;                                     % Free RAM.

%% Set the variable "vnamel" equal to "PROP_m" then multiply by
%% "Fshft_input" to form the output "Fshft_output."
    if m < 10,
      eval(['vnamel = PROP_0',int2str(m),';']);
    else
      eval(['vnamel = PROP_',int2str(m),';']);
    end
    Fshft_output = (vnamel .* (Fshft_input));
      %*% mesh(Fshft_output); title('FSHFT_OUTPUT ')      % Optional for
      %*% pause(2); clg;                        % viewing "Fshft_output."
    clear vnamel;                    % Make "vnamel" ready for next pass.
    eval(['clear PROP_0',int2str(m),';']);    % Clear unneeded variables
    eval(['clear PROP_',int2str(m),';']);                % to free RAM.

%% Shift "Fshft_output" to corner geometry prior to taking the inverse
%% 2-D Fourier transform, ifft2. Take the inverse transform to produce
%% "output." Then shift "output" back to the center geometry to produce
%% the diffracted wave at time slice "m" called "shft_output."
    F_output = fftshift(Fshft_output);
      %*% mesh(F_output); title('F_OUTPUT ')      % Optional for viewing
      %*% pause(2); clg;                                  % "F_output."
    clear Fshft_output;                                  % Free RAM.
    output =  (ifft2(F_output) );
```

```
      %*% mesh(output); title('OUTPUT '); % Optional for viewing
      %*% pause(2); clg;                  % "output."
    shft_output = fftshift(output);
      %*% mesh(shft_output); title('SHFT_OUTPUT '); % Optional for
      %*% pause(2); clg:                % viewing "shft_output."
    clear F_output; clear output;                  % Free RAM.

%% For optional view of the magnitude of the shifted output,
%% "shft_output."
    %*% mesh(abs(shft_output)); title('ABS(SHFT_OUTPUT)');
    %*% pause(2); clg;

%% Save the N0 (center) row of shifted output in the m-th column
%% of "output_plot." This creates an N x m+Step matrix whose columns
%% show the reduction in amplitude and the spreading of the wave.
    output_plot(1:N,m+Step) = (shft_output(N0,1:N))';

%% Save the N x N matrix "shft_output" in a file whose name is
%% "input_func"_OUT_"m".dat. A Gaussian, for example, would be
%% G_OUT_12.dat on the 12-th loop interation. This is optional for
%% graphics use by other software.
    %*% vname4 = ['shft_output',int2str(m)];
    %*% eval([vname4,'= shft_output ;']);
    %*% if m < 10,
      %*% eval(['save E:\',input_func,'_OUT_0',int2str(m),'.dat',
⇒              vname4,' /ascii']);
    %*% else
      %*% eval(['save E:\',input_func,'_OUT_',int2str(m),'.dat ',
⇒              vname4,' /ascii']);
    %*% end
    %*% eval(['clear shft_output ',vname4]);  % Get ready for next pass.
 end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% End loop %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Save the contents of "output_plot" in a MATLAB file and as an ascii
%% file.
 %*% filename = ['ACO',int2str(d),'x',int2str(m)];
 %*% eval(['save ',filename,' output_plot'] );
  eval(['save e:\',OutFile_Name,'.dat output_plot /ascii']);

%% An optional plot of three views of "output_plot."
 %*% subplot(121), mesh(rot90(output_plot,1));
 %*% subplot(122), mesh(rot90(output_plot,2));
 %*% pause(2); clg;
 %*% mesh(rot90(output_plot,3)),title('OUTPUT');
 %*% pause(3);
 %*% meta OUT_3                    % Optional, for producing a MATLAB
                                    % hard copy of the  third view.
```

```
%% Simultaneously display the input and output.
 subplot(224),
 mesh(rot90(abs(output_plot),3)),title('OUTPUT_absolute_value');
 pause(3)
%*% meta ABS_OUT                    % For producing a MATLAB hard copy.
```

APPENDIX F. SOURCE CODE FOR INPUT EXCITATIONS

The following source code is for the input excitation choices given to the user in AC_PROP. Each is written as a MATLAB function and can be used independently of AC_FIL or AC_PROP. The input excitations are the uniform square (TABLE(w,N)), the uniform circle (CRCLE(d,N)), the circularly truncated Gaussian (CRCGAUS(sigma,d,N)), and the circularly truncated Bessel (CRCBES(a,d,N)) where *w* is the width of the square, *d* is the diameter of the circle, *sigma* is the standard deviation of the Gaussian distribution, *a* is a scaling factor, and *N* is the size of the base array.

TABLE.M SOURCE CODE

```
function Y = table(w,N)
%     TABLE.M: Y = table(w,N)
%Program for generating a uniform square excitation function.
%   December 1992     William H. Reid
%   Based on TABLE.M by JG Upton
%
%   w is the WIDTH of the table.  (ODD integer)
%   N is the WIDTH of the square base.  (EVEN integer)
%   Example:  z = table(33,64);

% Check that w is an odd integer.
      if rem(w,2) < 0.1;
      error('The width of the table must be an ODD integer.');
      else;
      end;

% Check that N is an even integer.
      if rem(N,2) ~= 0.0;
      error('The width of the square base must be an EVEN integer.');
      else;
      end;
```

```
N0 = (N/2)+1;                          % N0 is the base array's center.
w0 = ceil(w/2);                        % w0 is the mid-point of the table.

Y = zeros(N);                          % Initialize matrices to reduce
temp = zeros(N0-1);                    % processing time.

temp(1:w0,1:w0) = ones(w0);            % Set amplitude to one.

% Generate the entire N x N input function array.
Y(N0:N,N0:N) = temp;
Y(2:N0,N0:N) = rot90(temp);
Y(2:N0,2:N0) = rot90(temp,2);
Y(N0:N,2:N0) = rot90(temp,3);

% To test input distribution:  mesh(Y)
```

CRCLE.M SOURCE CODE

```
function Y = crcle(d,N)
%     CRCLE.M:  Y = crcle(d,N)
% Program for generating uniform circular excitation functions
%   December 1992     William H. Reid
%   Based on CRCLE.M by JG Upton
%
%    d is the DIAMETER of the circle.  (ODD integer)
%    N is the WIDTH of the square base. (EVEN integer)
%    Example: z = crcle(33,64);

% Check that d is an odd integer.
     if rem(d,2) < 0.1;
     error('The diameter of the crcle function must be an ODD
⇒          integer.');
     else;
     end;

% Check that N is an even integer.
     if rem(N,2) ~= 0.0;
     error('The width of the square base must be an EVEN integer.');
     else;
     end;

N0 = (N/2)+1;                          % N0 is the base array's center.
r = d/2;                               % r is the circle's radius.

% Initialize matrices to reduce processing time.
Y = zeros(N);
temp = zeros(N0-1);
```

```
% Set amplitude to one inside the circle's radius.
for m = 1:r+1;
    for n = 1:r+1;
        if sqrt((m-1)^2 + (n-1)^2) <= r;
        temp(m,n) = 1;
        end;
    end;
end;

% Generate the entire N x N input function.
Y(N0:N,N0:N) = temp;
Y(2:N0,N0:N) = flipud(temp);
Y(2:N0,2:N0) = rot90(temp,2);
Y(N0:N,2:N0) = fliplr(temp);

% To test input function distribution:  mesh(Y)
```

CRCGAUS.M SOURCE CODE

```
function Y = crcgaus(sigma,d,N)
%     CRCGAUS.M:  Y = crcgaus(sigma,d,N)
% Program for generating circular Gaussian excitation functions.
%   December 1992     William H. Reid
%   Based on CRCGAUS.M by JG Upton
%
%  sigma is the STANDARD DEVIATION of the gaussian function.
%  d is the  DIAMETER of circle.  (ODD integer)
%  N is the WIDTH of the square base. (EVEN integer)
%  Example: z = crcgaus(12,33,64);

mu=0;                          %mu is the mean of the gaussian function.

% Check that d is an odd integer.
      if rem(d,2) < 0.1;
      error('The diameter of the circle function must be an ODD
⇒           integer.');
      else;
      end;

% Check that N is an even integer.
      if rem(N,2) ~= 0.0;
      error('The width of the square base must be an EVEN integer.');
      else;
      end;

N0 = (N/2)+1;                  % N0 is center of the array.
r = d/2;                       % r is the radius of the truncating circle.
```

```
% Initialize the matrices to reduce processing time.
Y = zeros(N);
temp = zeros(N0-1);

% Compute the amplitude for the Gaussian distributed circle.
for m = 1:(d+1)/2;
    for n = 1:(d+1)/2;
        x = sqrt((m-1)^2+(n-1)^2);
        if x <= r;
        temp(m,n) = (1/(sqrt(2*pi)*sigma))*exp(-((x-mu)^2)/...
                    (2*(sigma^2)));
        end;
    end;
end;

% Generate the entire N x N input array.
Y(N0:N,N0:N) = temp;
Y(2:N0,N0:N) = flipud(temp);
Y(2:N0,2:N0) = rot90(temp,2);
Y(N0:N,2:N0) = fliplr(temp);

Y = Y ./ (max(max(Y)));  % Normalize the Gaussian distribution to one.

% To test and view the input function:  mesh(Y)
```

CRCBESS.M SOURCE CODE

```
function Y = crcbess(a,d,N)
%     CRCBESS.M:  Y = crcbess(a,d,N)
% Program for generating circular Bessel excitation functions.
%   December 1992     William H. Reid
%   Based on CRCBESS.M by JG Upton
%
%    a is the WIDTH SCALING FACTOR.
%    d is the DIAMETER of the circle.  (ODD integer)
%    N is the WIDTH of the square base. (EVEN integer)
%    Example: z = crcbess(1,33,64);

% Check that d is an odd integer.
       if rem(d,2) < 0.1;
       error('The diameter of the circle must be an ODD integer');
       else;
       end;

% Check that N is an even integer.
       if rem(N,2) ~= 0.0;
       error('The width of the square base must be an EVEN integer');
       else;
```

```
       end;

N0 = (N/2)+1;                          % N0 is the center of the array.
r = d/2;                               % r is the radius of the circle.

Y = zeros(N);                          % Initialize the arrays to reduce
temp = zeros(N0-1);                    %  processing time.

% Compute the Bessel distributed amplitude within the circle.
for m = 1:r+1;
    for n = 1:r+1;
        x = sqrt((m-1)^2 + (n-1)^2);
        if x <= r;
        temp(m,n)=besseln(0,a*x);
        end;
     end;
 end;

% Generate the entire N x N input array.
Y(N0:N,N0:N) = temp;
Y(2:N0,N0:N) = flipud(temp);
Y(2:N0,2:N0) = rot90(temp,2);
Y(N0:N,2:N0) = fliplr(temp);

% To test and view the input function:  mesh(Y)
```

APPENDIX G. EXAMPLES OF OUTPUT FROM A GAUSSIAN INPUT

This appendix contains examples of the outputs that result from Gaussian inputs. The Gaussian inputs in all cases have been normalized. The outputs give the magnitude as a function of time and radial distance. Each output represents only $1/N$ of the data computed; this is done so that a four dimensional field (the four dimensions are x, y, z, and *time*) can be represented in three dimensions (the magnitude for all x values, radial distance values, at $y = 0$ and $z = 10$ cm as a function of *time*). Each output for a given diameter d differs in the standard deviation σ of the input. Two diameters, 51 and 25 samples, are provided with respective metric diameters of 3.88 cm and 1.86 cm. In each diameter case, the first input/output combination is the Circle function of the same diameter.

The base array size for all figures of this appendix is 128x128. Titling of the images is of the form G{input/output indicator}σxd where the input and output indicators are underscore _ and O, respectively. The images are in order of decreasing σ.

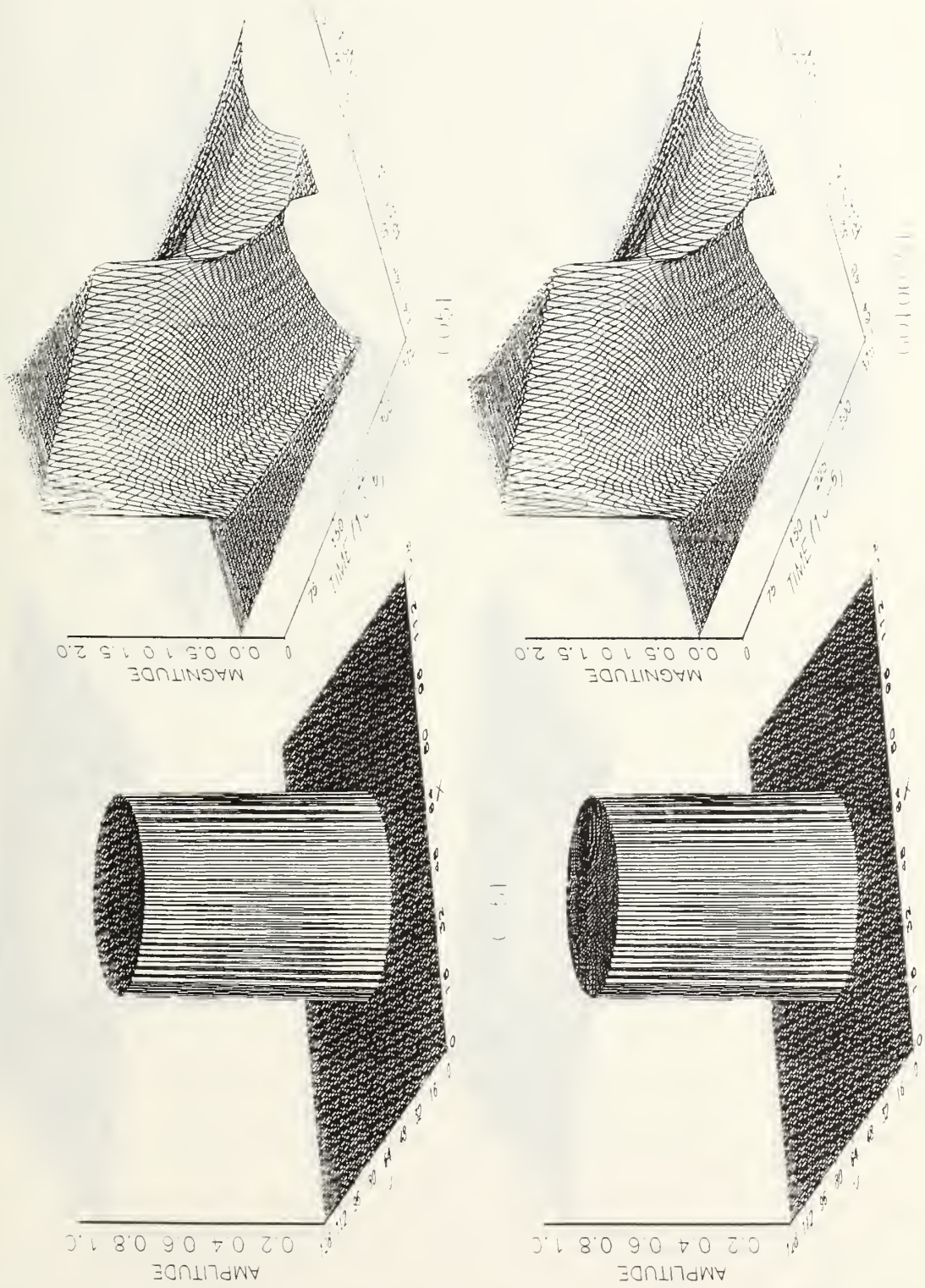

Figure 15. **Circle** with $d = 51$ and **Gaussian** with $\sigma = 100$ inputs and outputs.

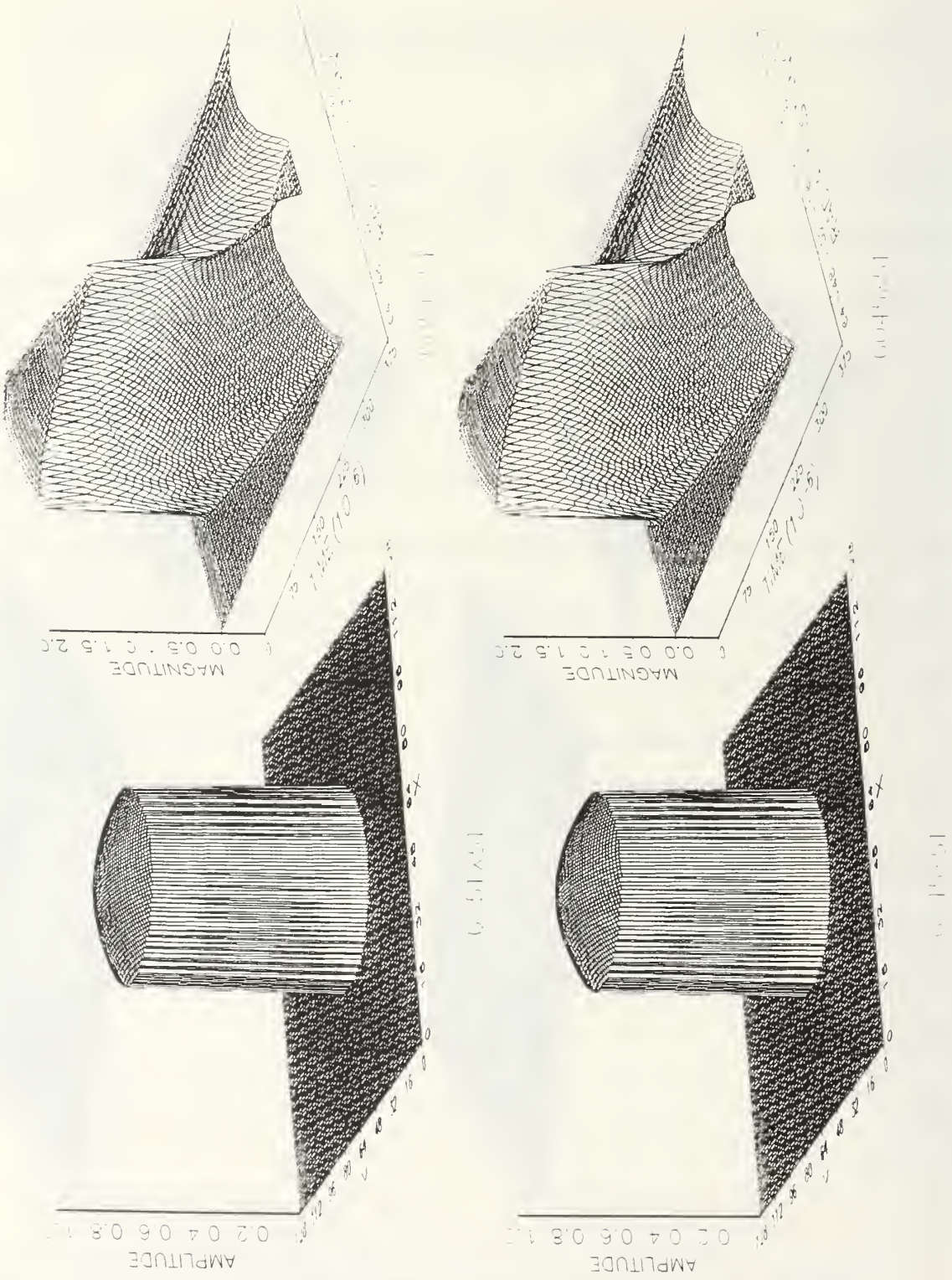

Figure 16. Example **Gaussian** inputs and outputs for $\sigma = 51$ and $\sigma = 45$.

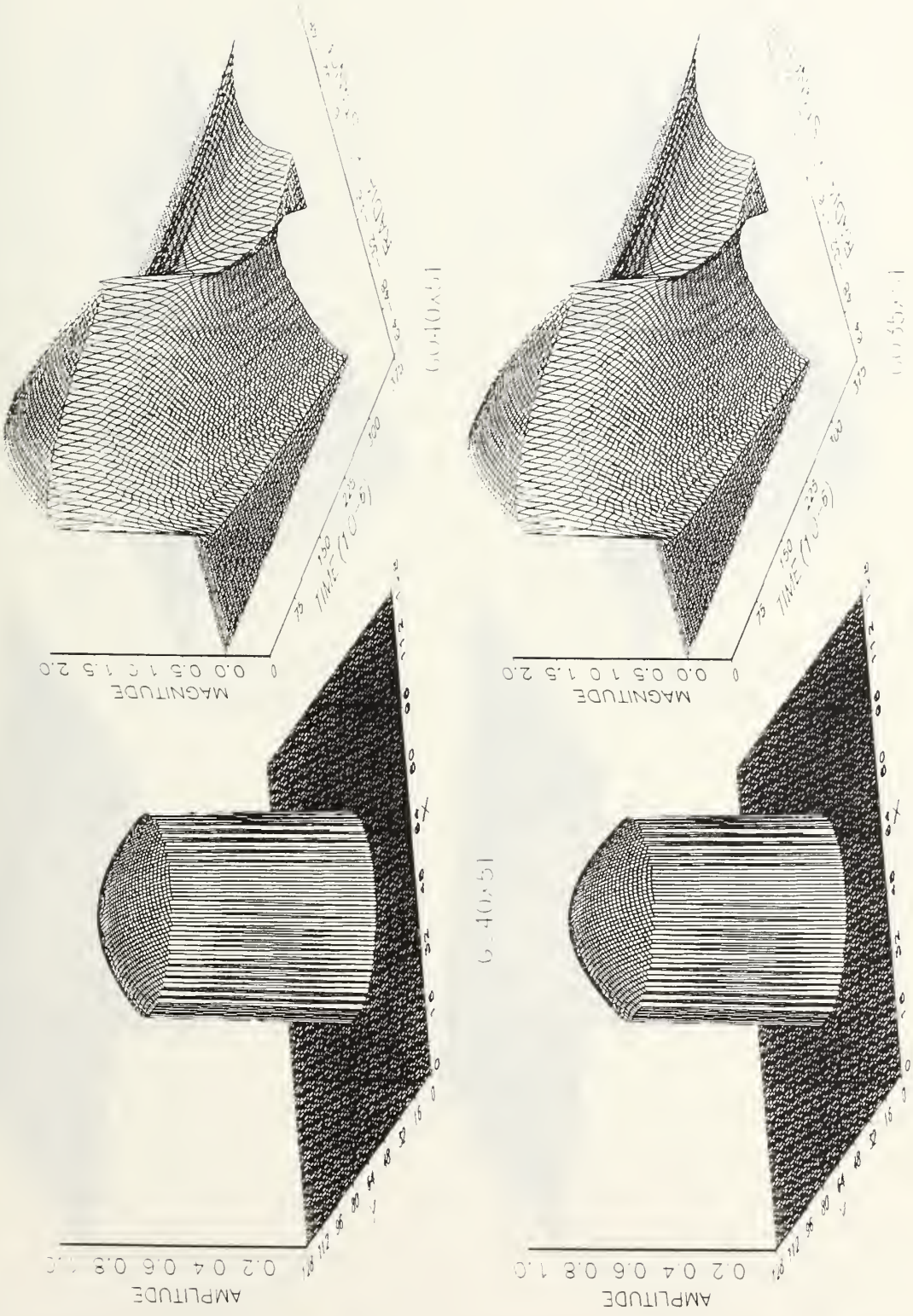

Figure 17. Example **Gaussian** inputs and outputs for σ = 40 and σ =35.

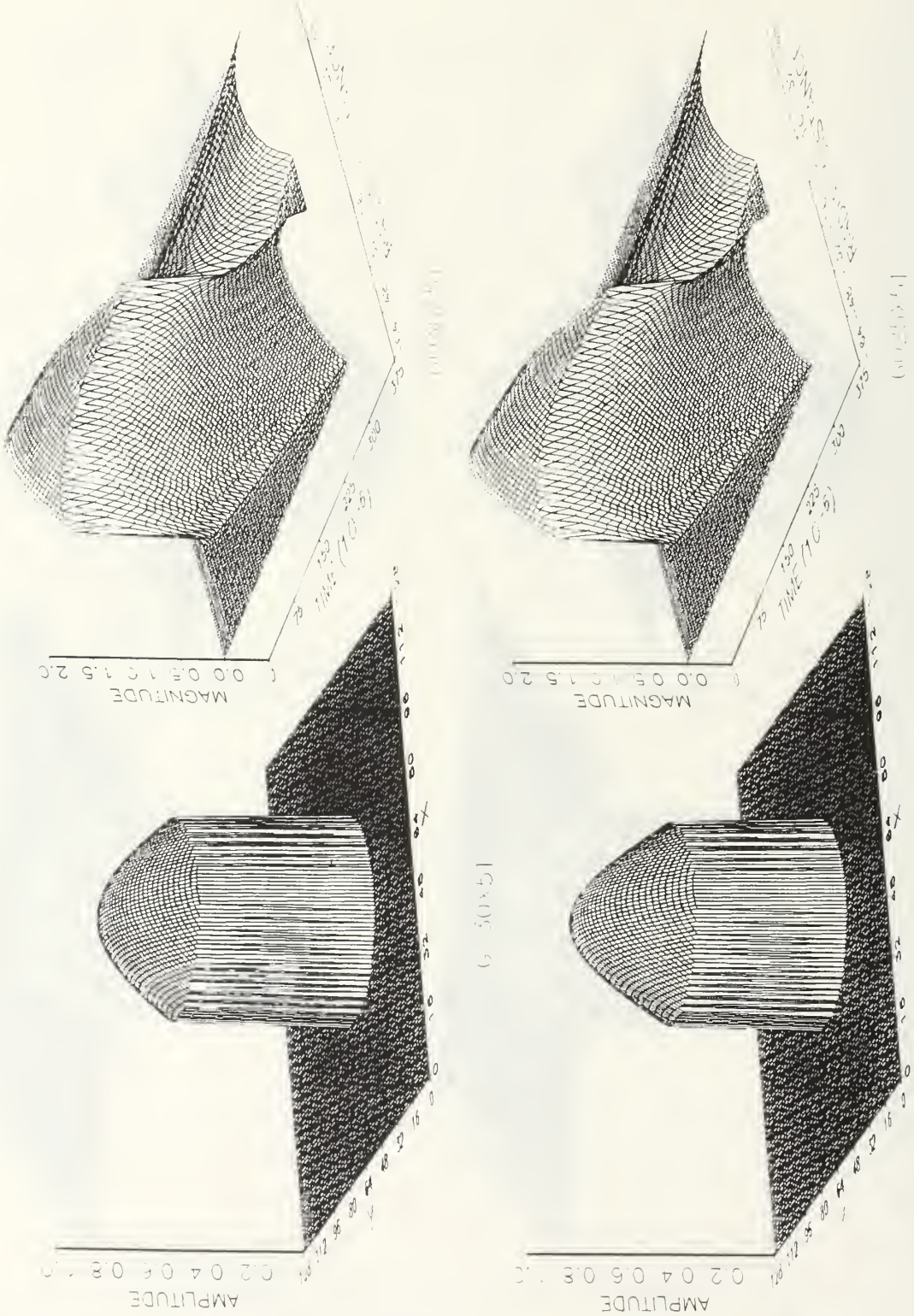

Figure 18. Example **Gaussian** inputs and outputs for $\sigma = 30$ and $\sigma = 25$.

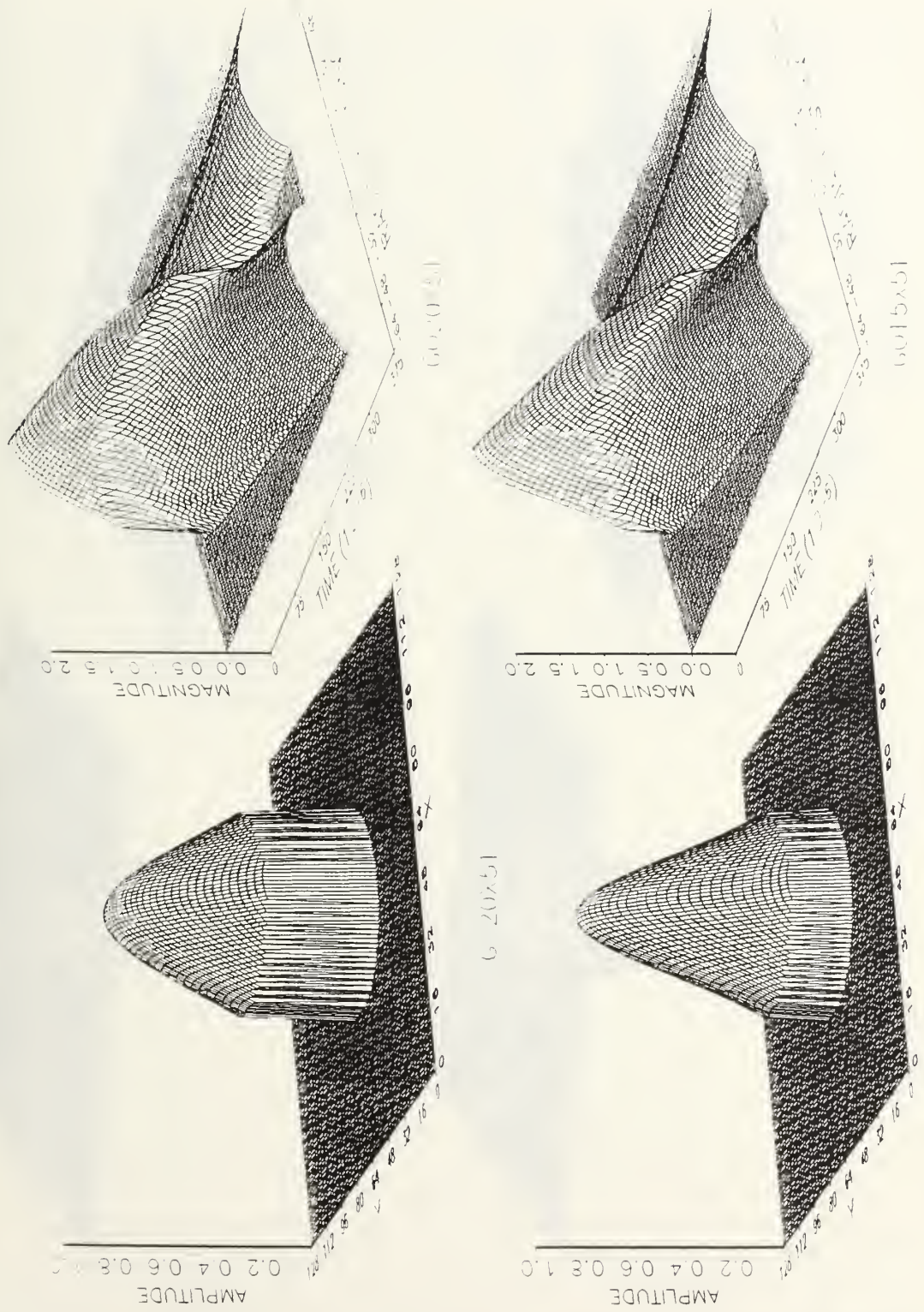

Figure 19. Example **Gaussian** inputs and outputs for $\sigma = 20$ and $\sigma = 15$.

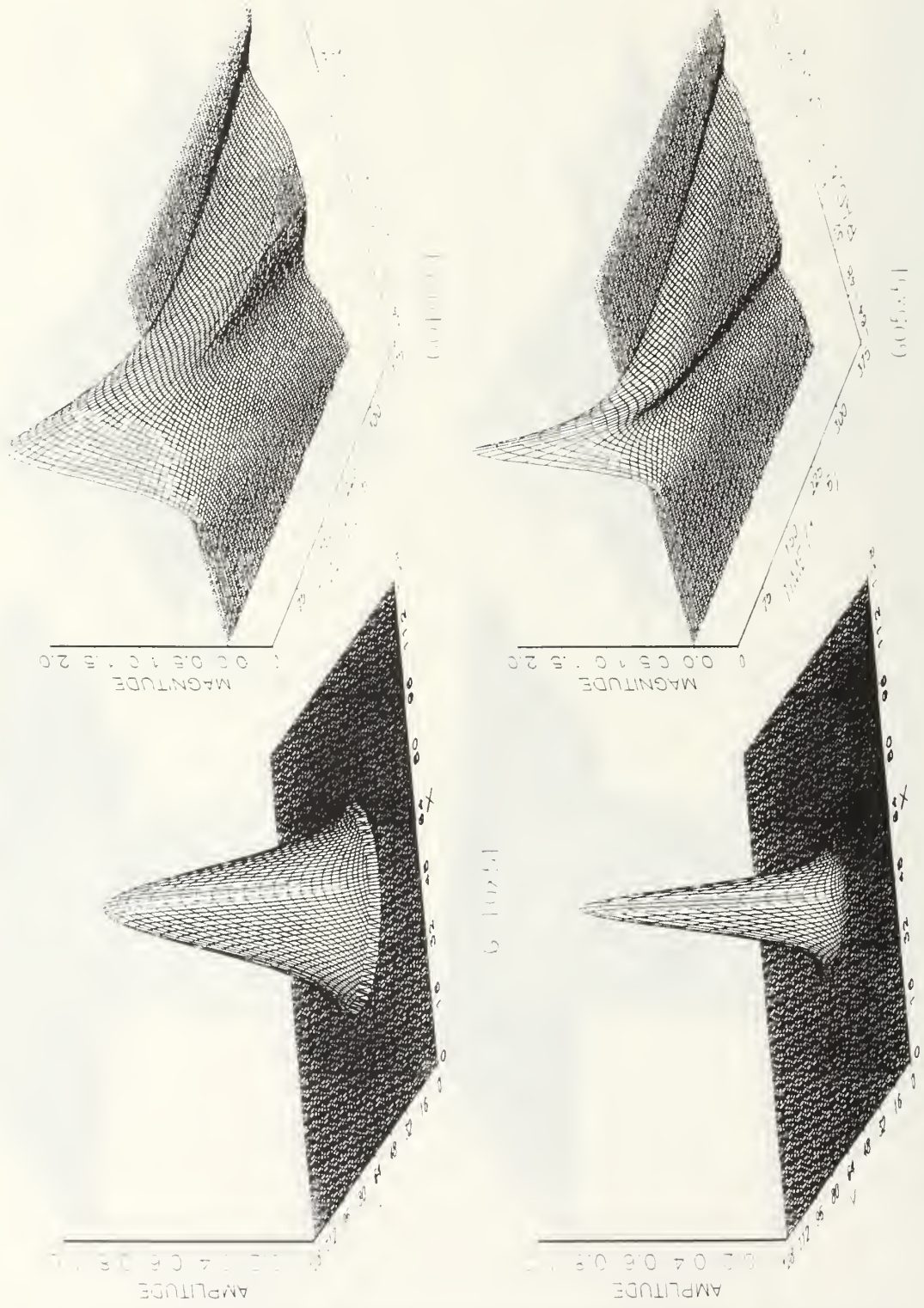

Figure 20. Example **Gaussian** inputs and outputs for $\sigma = 10$ and $\sigma = 5$.

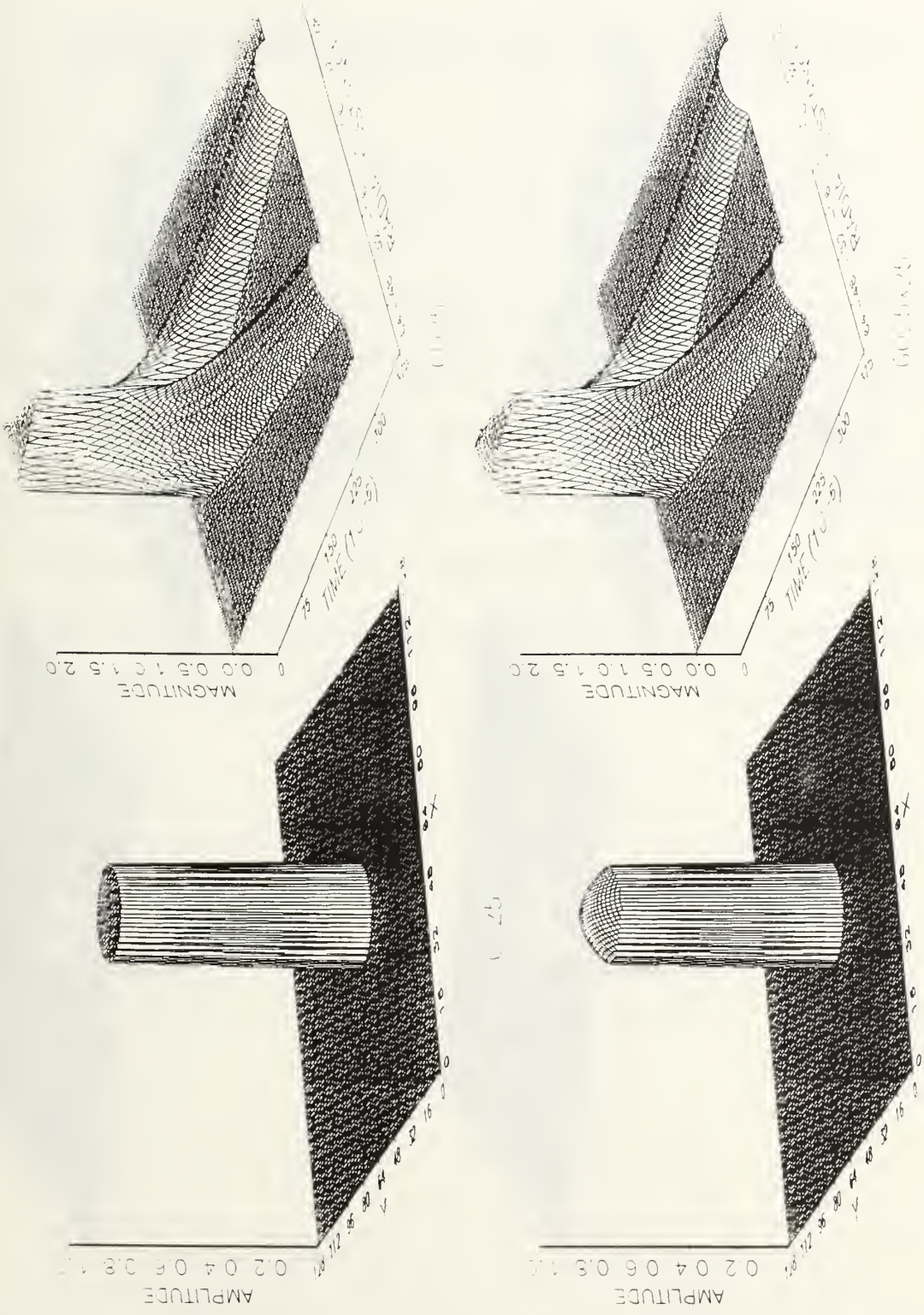

Figure 21. **Circle** with $d = 25$ and **Gaussian** with $\sigma = 25$ inputs and outputs.

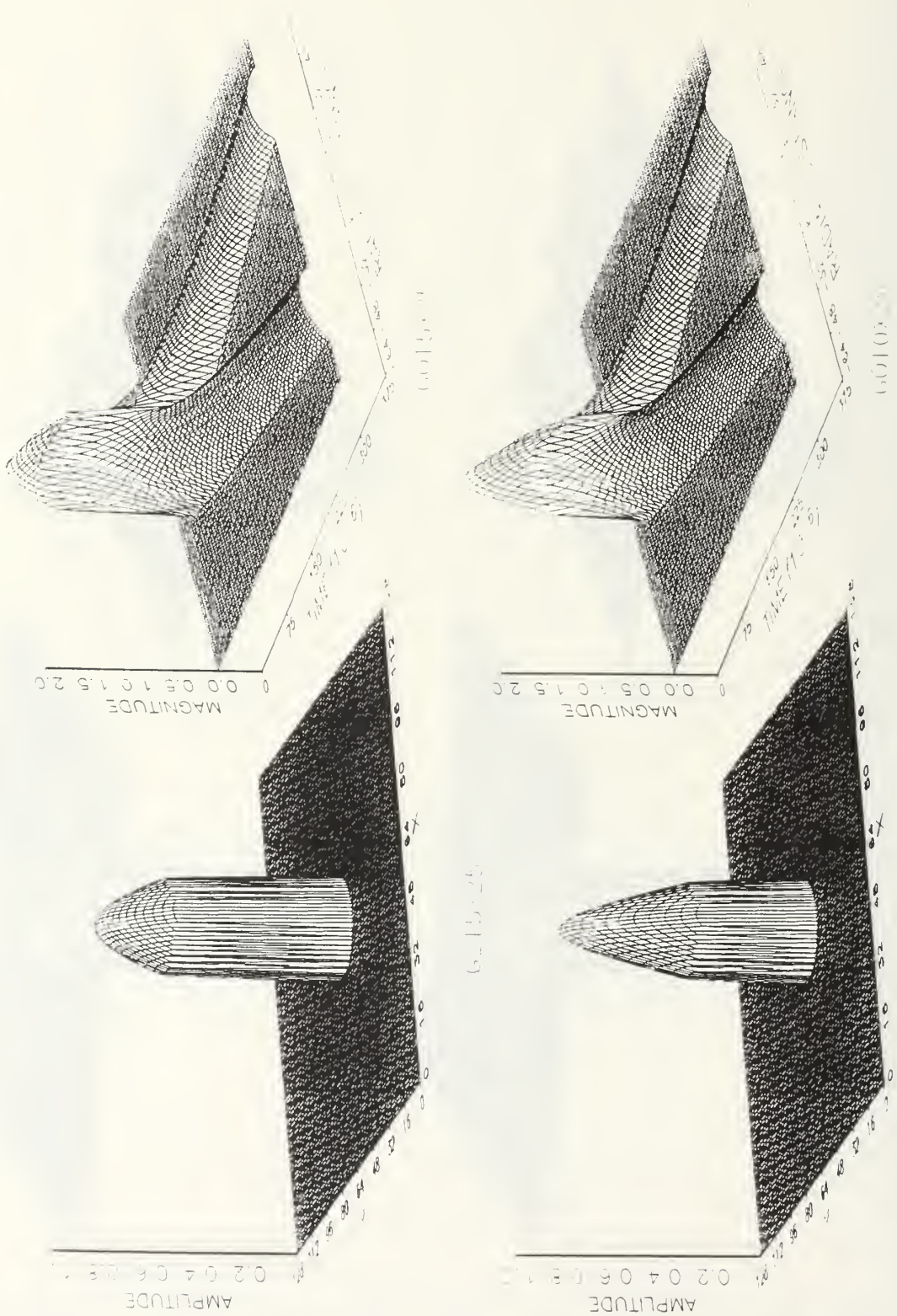

Figure 22. Example **Gaussian** inputs and outputs for $\sigma = 15$ and $\sigma = 10$.

APPENDIX H. EXAMPLES OF OUTPUT FROM A BESSEL INPUT

This appendix contains examples of the Bessel input excitations and the resulting outputs. The inputs are 128x128 time domain array representations. The resulting outputs are $1/128^{th}$ of the data computed; this is done so that a four dimensional field (the four dimensions are x, y, z, and *time*) could be represented on a three dimensional plot (the magnitude for all x values, radial distance values, at $y = 0$ and $z = 10$ cm as a function of *time*). Two cases of diameter are shown, 51 and 25, having metric equivalents of 3.88 cm and 1.86 cm. Titling of the images is as discussed in the thesis body; however, a summary of the titling is given in Table IV for convenience.

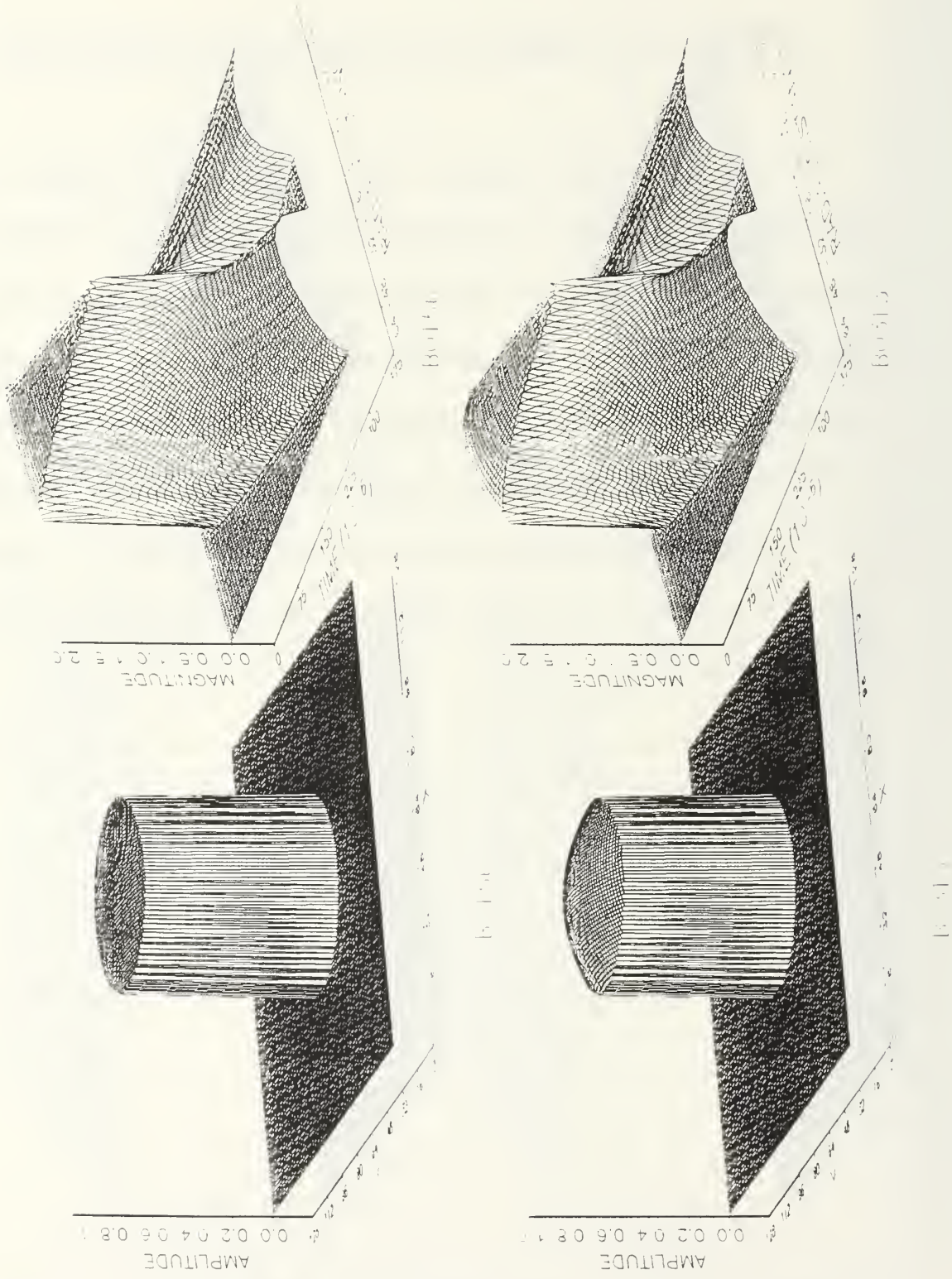

Figure 23. **Bessel** inputs and outputs for $a = 1/64$ (top) and $a = 1/32$ (bottom).

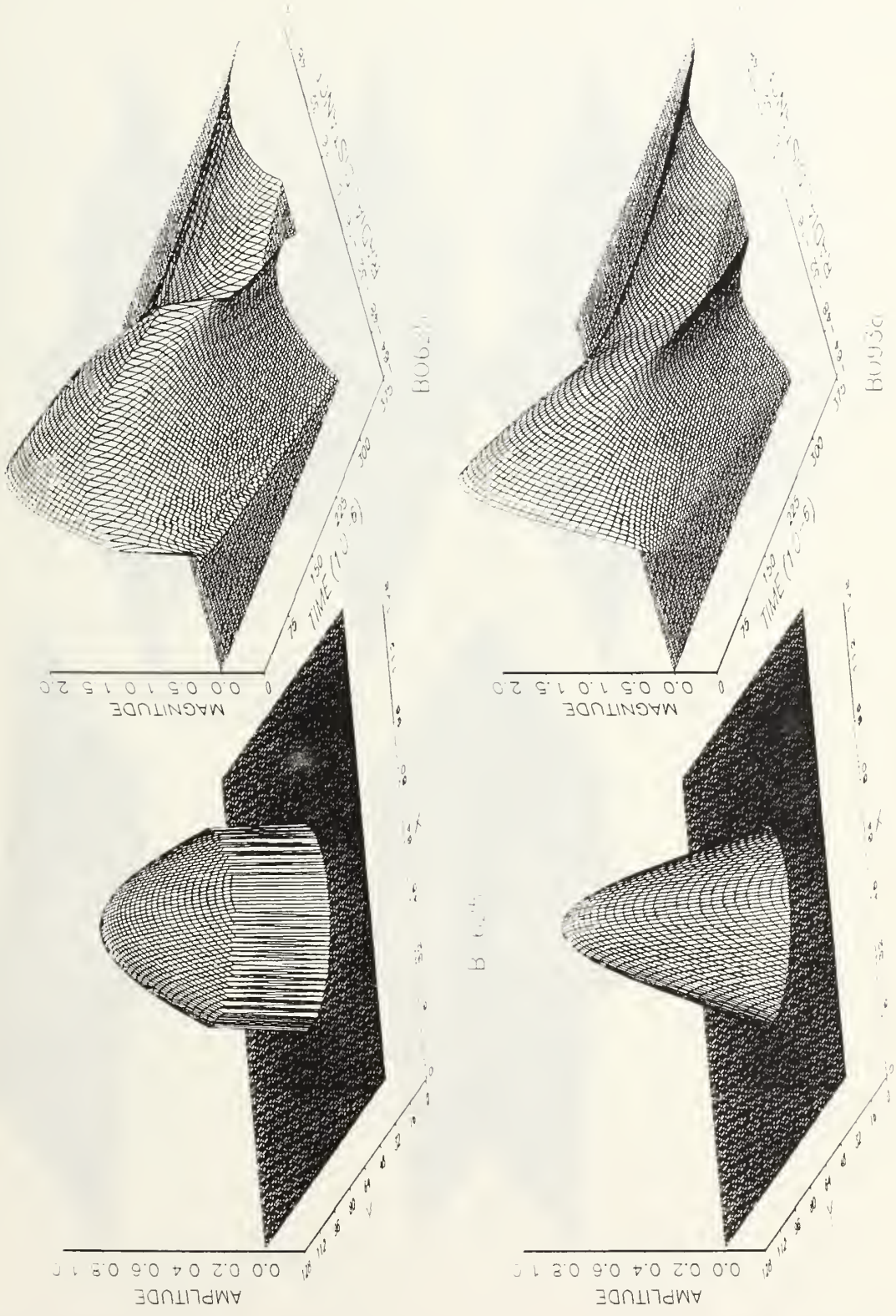

Figure 24. **Bessel** inputs and outputs for $a = 1/16$ (top) and $a = 3/32$ (bottom).

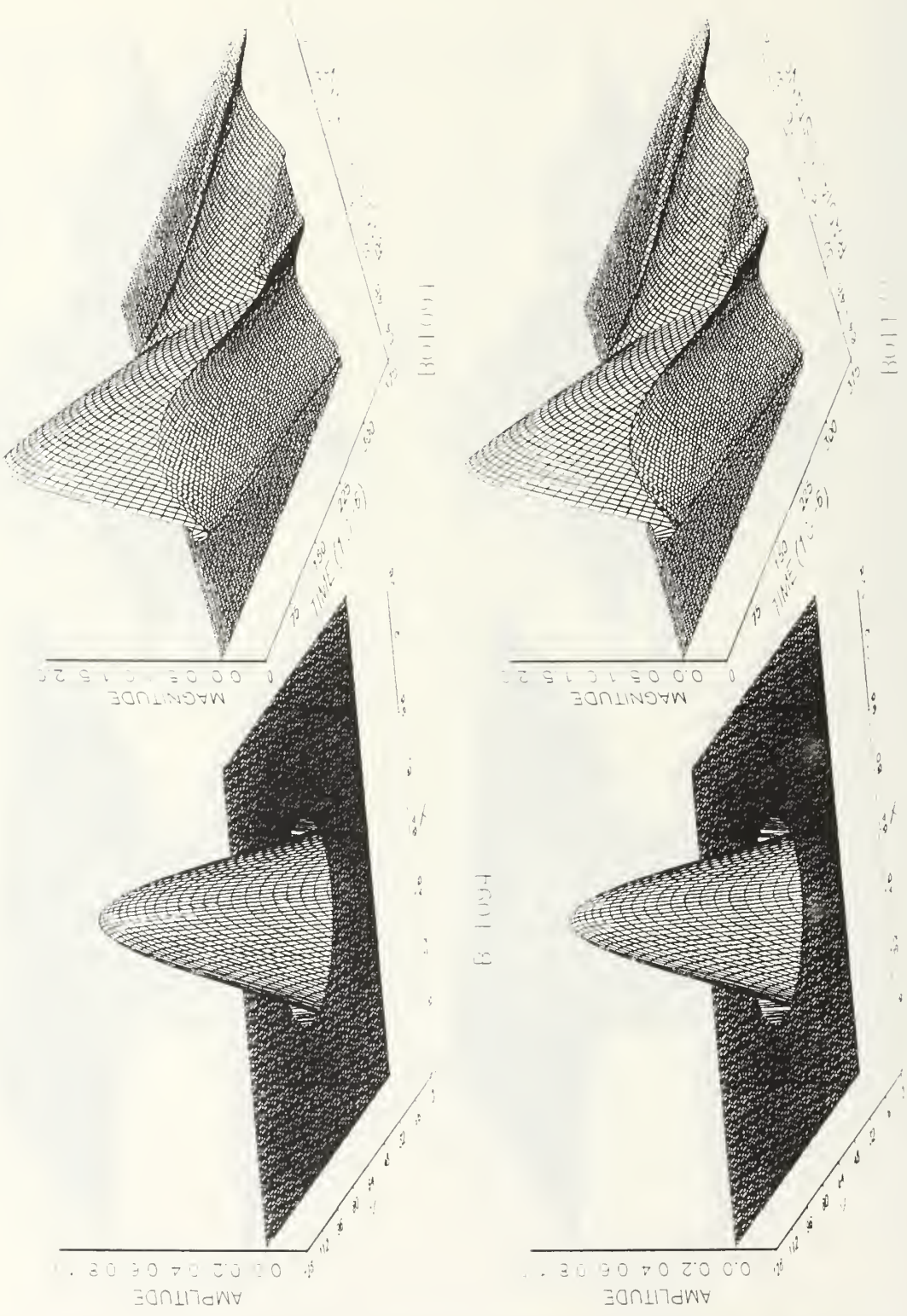

Figure 25. **Bessel** input and outputs for $a = 7/64$ (top) and $a = 15/128$ (bottom).

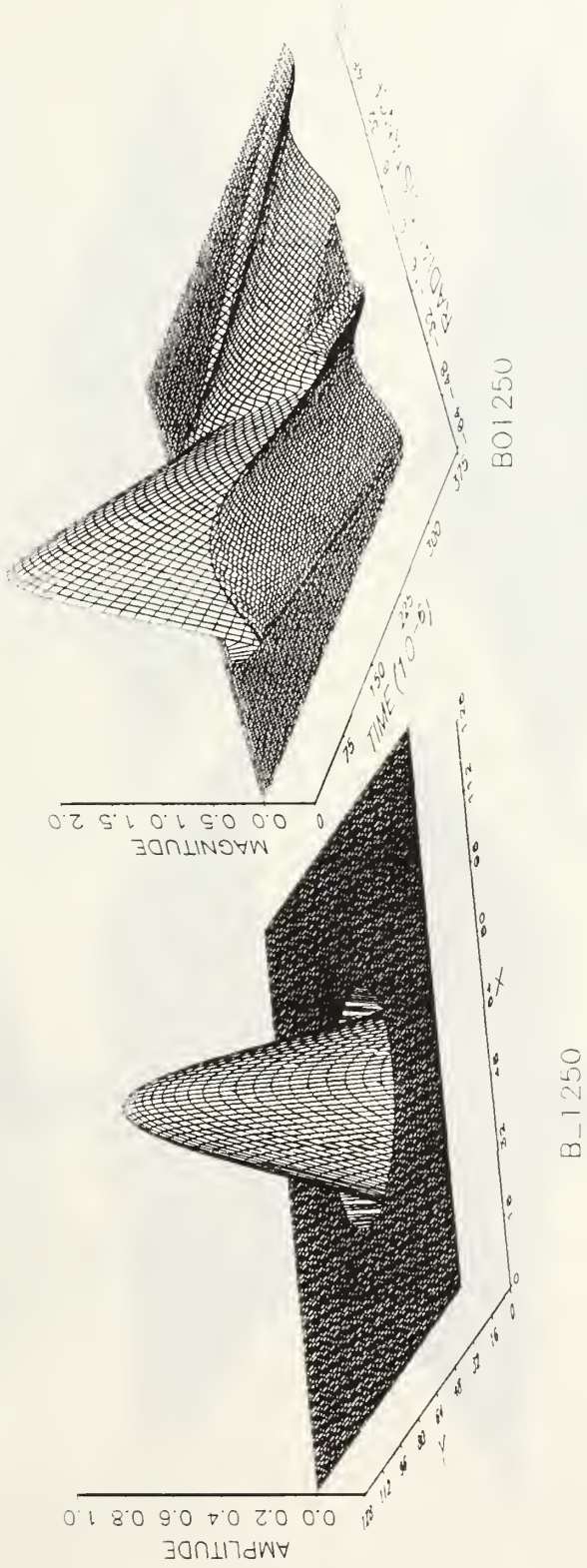

Figure 26. **Bessel** input and output for $a = 1/8$.

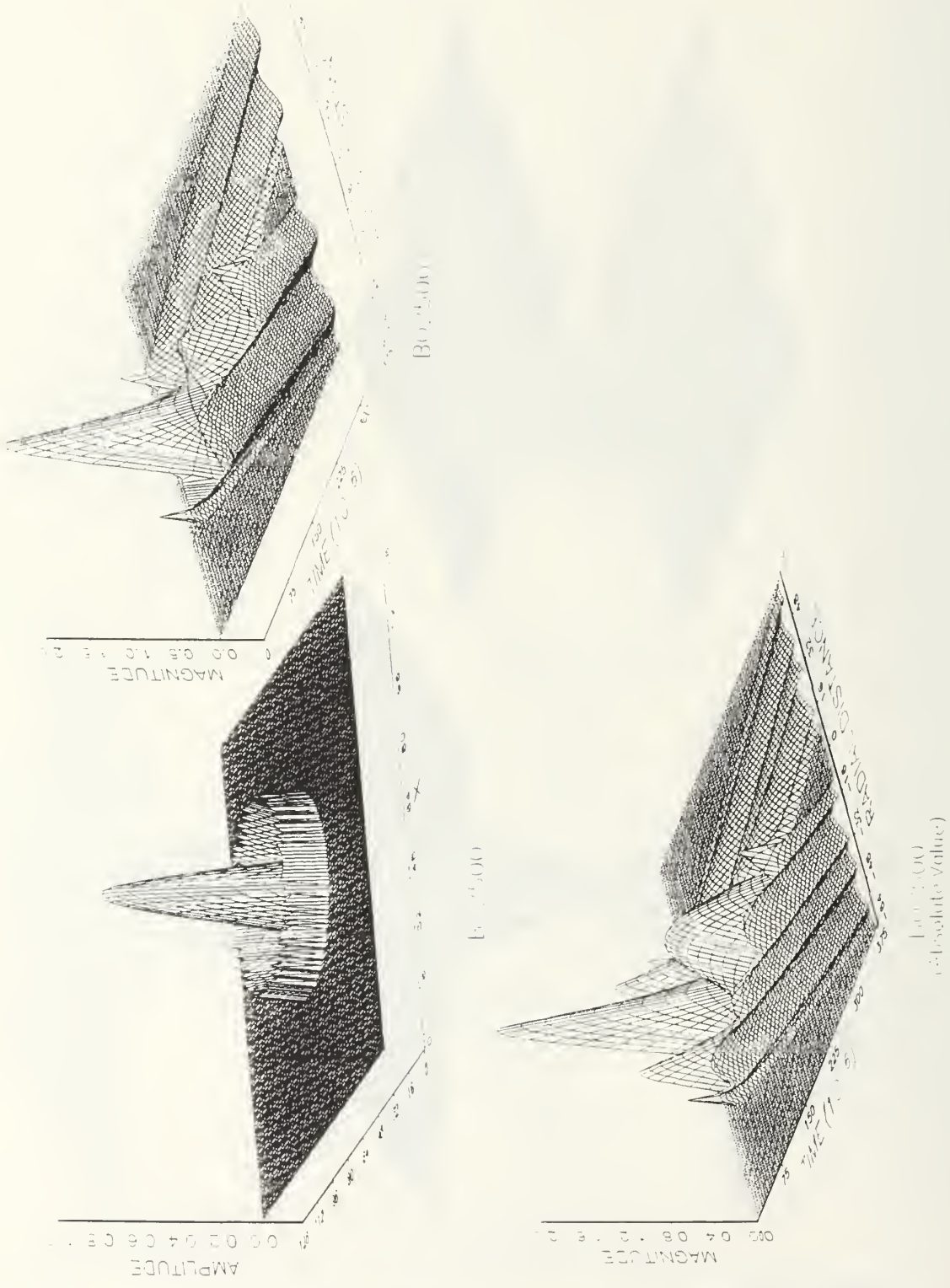

Figure 27. **Bessel** input for $a = 1/4$ and two output formats.

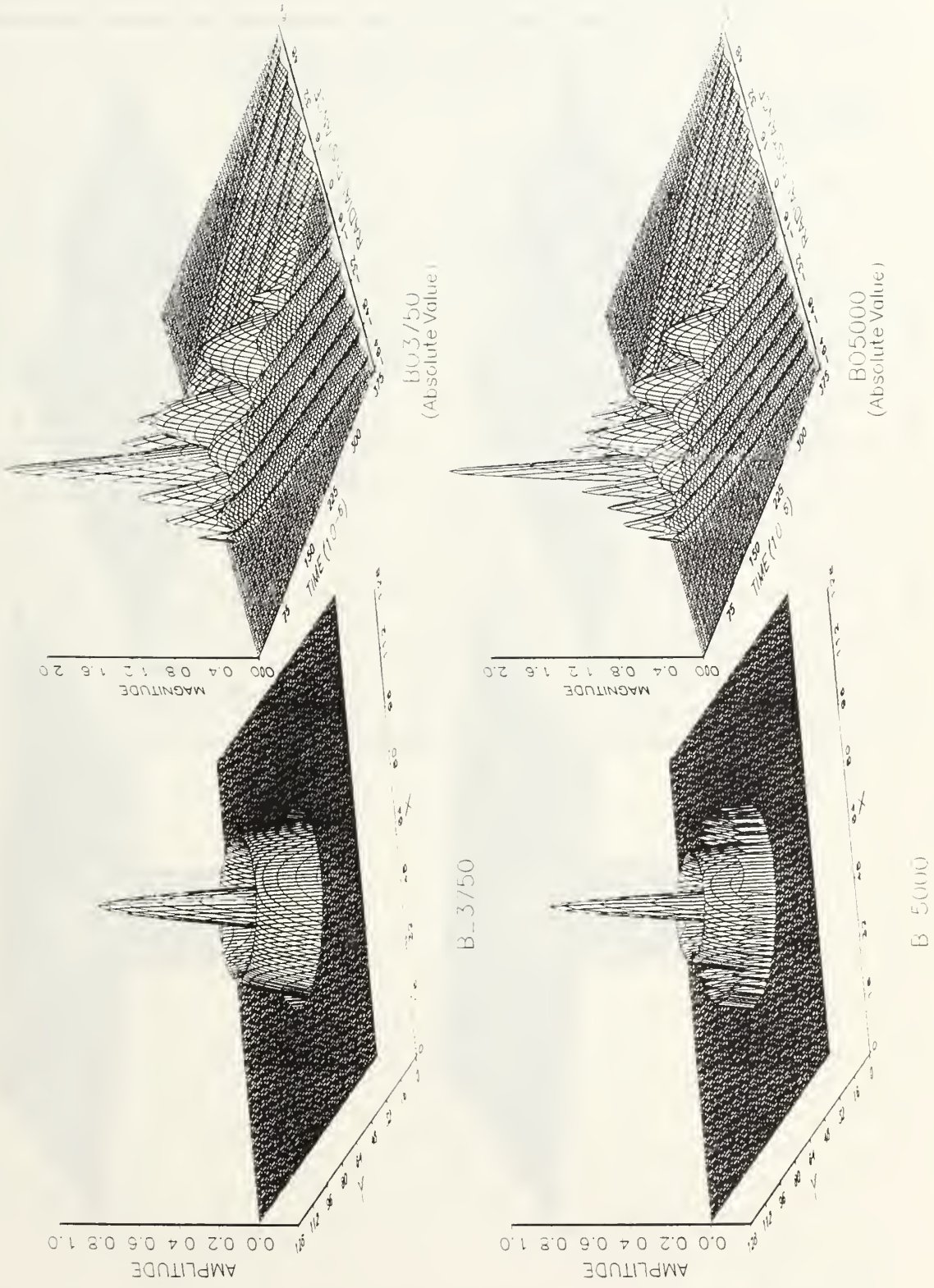

Figure 28. **Bessel** inputs and outputs for $a = 3/8$ (top) and $a = 1/2$ (bottom).

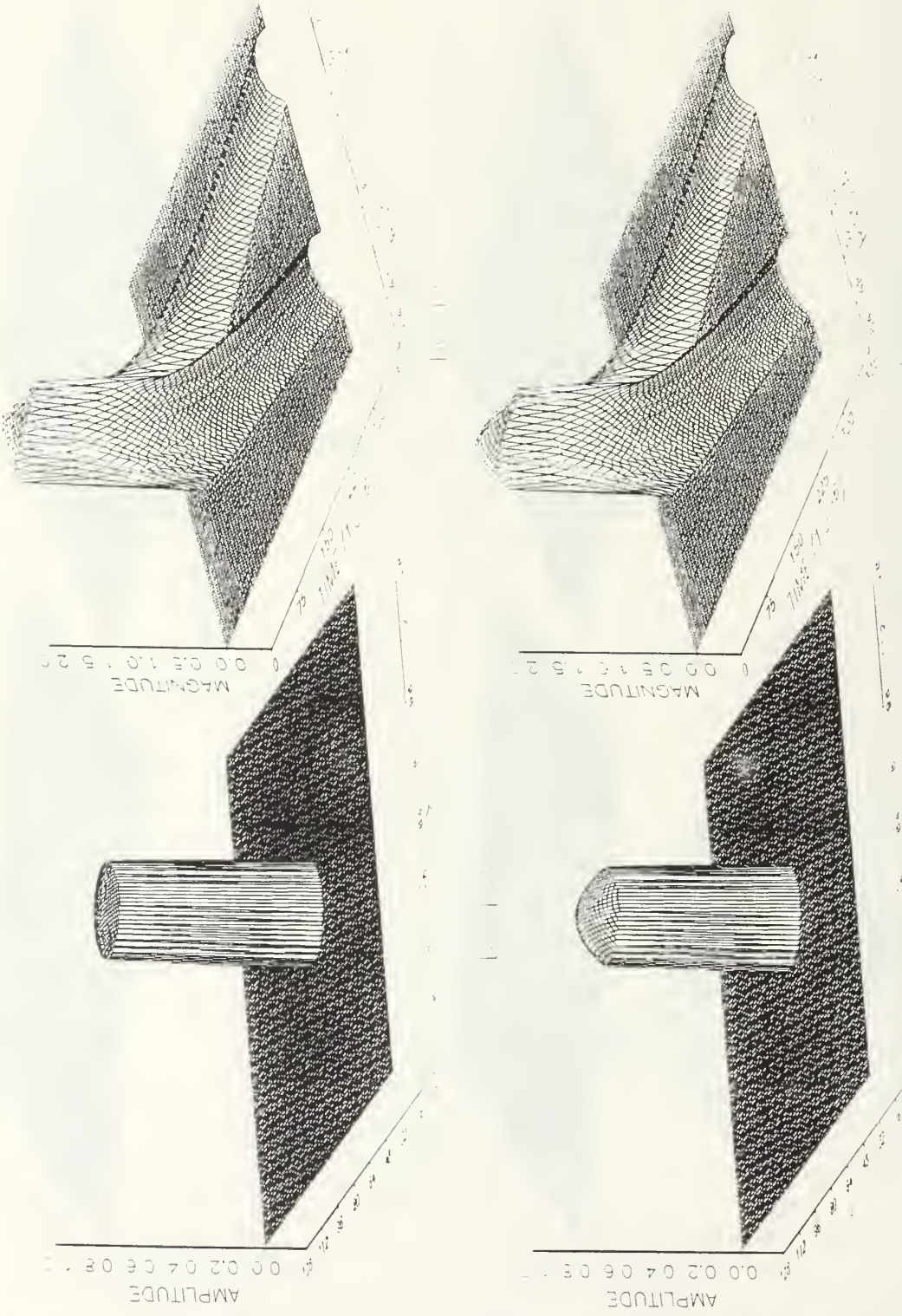

Figure 29. **Bessel** inputs and outputs for $d = 25$, $a = 1/32$ (top) and $1/16$ (bottom).

Table VI. KEY TO LABELING OF **BESSEL** EXAMPLE GRAPHICS.

BESSEL EXCITATION EXAMPLES			
NAME	VALUE OF a	SUMMARY	FIGURE NUMBER
B_156, BO156	1/64	INPUT, OUTPUT	23
B_313, BO313	1/32	INPUT, OUTPUT	23
B_625, BO625	1/16	INPUT, OUTPUT	24
B_938, BO938	3/32	INPUT, OUTPUT	24
B_1094, BO1094	7/64	INPUT, OUTPUT	25
B_1172, BO1172	15/128	INPUT, OUTPUT	25
B_1250, BO1250	1/8	INPUT, OUTPUT	26
B_2500, BO2500	1/4	INPUT, OUTPUT	27
B_3750, BO3750	3/8	INPUT, OUTPUT	28
B_5000, BO5000	1/2	INPUT, OUTPUT	28
B_313A, BO313A	1/32, d = 25	INPUT, OUTPUT	29
B_625A, BO625A	1/16, d = 25	INPUT, OUTPUT	29

LIST OF REFERENCES

1. J. W. Goodman, Introduction to Fourier Optics. McGraw-Hill, Inc., San Francisco, CA: 1968.

2. P. R. Stepanishen, "Transient radiation from pistons in an infinite planar baffle," Journal of Acoustical Society of America, vol. 49, no. 5, pp. 1629-1637 (1971).

3. P. R. Stepanishen, "Acoustic transients in the far-field of a baffled circular piston using the impulse response approach," Journal of Sound and Vibration, vol. 32, no. 3, pp. 295-310 (1974).

4. P. R. Stepanishen, "Acoustic transients from planar axisymmetric vibrators using the impulse response method," Journal of Acoustical Society of America, vol. 70, no. 4, pp. 1176-1181 (1981).

5. P. R. Stepanishen and G. Fisher, "Experimental verification of the impulse response method to evaluate transient acoustic fields," Journal of Acoustical Society of America, vol. 69, no. 6, pp. 1610-1617 (1981).

6. G. R. Harris, "Review of transient field theory for a baffled planar piston," Journal of Acoustical Society of America, vol. 70, no. 1, pp. 10-20 (1981).

7. D. Guyomar and J. Powers, "A Fourier approach to diffraction of pulsed ultrasonic waves in lossless media," Proceedings of the 1985 IEEE Ultrasonics Symposium, B. R. McAvoy, Ed.: IEEE Press, New York, 1985, pp. 692-695.

8. D. Guyomar and J. Powers, "Boundary effects on transient radiation fields from vibrating surfaces," Journal of Acoustical Society of America, vol. 77, no. 3, pp. 907-915 (1985).

9. D. Guyomar and J. Powers, "A Fourier approach to diffraction of pulsed ultrasonic waves in lossless media," Journal of Acoustical Society of America, vol. 82, no. 1, pp. 354-359 (1987).

10. D. Guyomar and J. Powers, "A transfer function model for propagation in homogeneous media," International Symposium on Pattern Recognition and Acoustical Imaging, Proc. SPIE 768, Bellingham, WA: pp. 253-257 (1987).

11. T. Merrill, "A Transfer Function Approach to Scalar Wave Propagation in Lossy and Lossless Media," Master's Degree Thesis, Naval Postgraduate School, March 1987.

12. MATLAB for MS-DOS Personal Computers, User's Guide by The Mathworks, Inc., Natick, MA (1990).

13. AXUM, Technical Graphics and Data Analysis, User's Manual by TriMetrix, Inc., Seattle, WA (1989).

14. Jian-yu Lu and James F. Greenleaf, "Nondiffracting X waves—exact solutions to free-space scalar wave equation and their finite aperture realizations," IEEE Transactions on Ultrasonic, Ferroelectrics, and Frequency Control, vol. 39, no. 1, pp. 19-31 (1992).

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2. Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3. Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5002	1
4. Professor John P. Powers, Code EC/Po Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5002	4
5. Professor Ron J. Pieper, Code EC/Pr Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5002	1
6. Lt. William H. Reid, USN 1502 Rivers Street Pensacola, Florida 32514	2

Thesis
R3254 Reid
c.1 Microcomputer simulation of a Fourier approach to ultrasonic wave propagation.